Department of Informatics

Decentralized Multi-resource Allocation in Clouds

Dissertation submitted to the
Faculty of Business, Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor / Doktorin der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, Ph.D.)

presented by
Patrick Gwydion Poullie
from
Germany

approved in September 2017

accepted on the recommendation of
Prof. Dr. Burkhard Stiller,
Ioanna Papafili, Ph.D.,
Prof. Georgios Stamoulis, Ph.D.

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, September 20th, 2017

Chairwoman of the Doctoral Board: Prof. Dr. Elaine M. Huang

Abstract

Cloud computing is omnipresent nowadays, as it allows for a fine-grained partitioning of data center resources and for flexibly providing access to these resources to any device that is connected to the Internet. This partitioning and provision of resources is achieved by hosting several Virtual Machines (VM) on the same physical machine. These VMs are rented out to customers to whom they look as if they were physical machines. Due to this ability to commercially rent out data center resources in a flexible, efficient, and fine-grained manner, cloud computing is deployed by virtually all companies and many private customers. However, commercial clouds are not always able to satisfy privacy and performance concerns of customers or they do not comply with regulations. Therefore, many companies and institutions also operate a private cloud, allowing them to complement VMs rented from commercial clouds with VMs hosted in their private cloud. In contrast to commercial clouds, the performance of VMs in a private cloud is not captured by Service Level Agreements, and therefore, all VMs are treated as processes of equal importance. As users operate different numbers of VMs and these utilize different amounts of physical resources, this equal treatment of VMs leads to users receiving unequal amounts of physical resources.

Thus, this thesis improves this situation by defining an efficient approach to enforce fairness in private clouds. An investigation of steps that practically allocate data center resources reveals that cloud resources are effectively controlled by changing priorities of VMs to access physical resources of their host, subsequently termed prioritization of VMs. Additionally, it is shown experimentally that no assumptions on respective utility functions can be made, when prioritizing VMs. These findings stand in contrast to the majority of work on fair data center resource allocations, which suggests the enforcement of fairness via scheduling and makes strong assumptions on utility functions. Realistically, well-defined utility functions or a well-structured

negotiation process cannot be assumed for the prioritization of VMs, thus, hosts are conceptualized as "self-serving buffets".

The premiss of this thesis that it is fair to constrain greedy consumers in favor of less greedy consumers in such a self-serving buffet, requires a metric that quantifies the greediness of consumers based on their multi-resource self-servings from a shared resource pool. Accordingly, the Greediness Metric is developed based on a questionnaire among more than 600 participants on the intuitive understanding of greediness and fairness. This Greediness Metric is formulated, evaluated, and outperforms all commonly known metrics to quantify multi-resource self-servings.

Based on a novel, formal cloud model, the Greediness Metric is refined to quantify the greediness of cloud users. Thus, greediness fairness is defined as the procedure of prioritizing VMs inversely to the greediness of their users. Under idealistic assumptions, enforcing greediness fairness provides sharing incentive, Pareto efficiency, strategy proofness, and envy freeness, just as Dominant Resource Fairness (DRF) does, which determines the state-of-the-art of fair data center multi-resource allocations. However, in contrast to DRF, enforcing greediness fairness based on the new Greediness Metric also provides incentives to users (a) to configure their VMs correctly, (b) to do that to the best of their knowledge in case of uncertainty, and (c) to neither unnecessarily partition workloads to multiple small VMs nor to concentrate workloads to one monolithic VM. A simulative investigation outlines how enforcing greediness fairness coordinates the resource allocation among nodes and improves the fairness among users. Furthermore, the simulative investigation shows that deploying the refined Greediness Metric to prioritize VMs results in an allocation that lies between the allocations generated, when using the metric used by DRF and the straight-forward metric of summing up resource amounts. Thus, the refined Greediness Metric is considered superior to prioritize VMs, as it leads to intuitive fairness among users and provides incentives to them to configure their VMs correctly.

To demonstrate the practical applicability of this approach, OpenStack is extended by a prototypical nova service called "nova-fairness" that enforces greediness fairness according to the new metric and is compatible with most of the known hypervisors. The processing overhead of the nova-fairness is evaluated in experiments and it is proven that it enforces fairness among users by coordinating the VM prioritization on hosts. This is combined with a messaging scheme that makes the nova-fairness highly scalable.

In summary, this thesis defines intuitive multi-resource fairness without access to any consumers' utility functions. The respective fairness definition is applied to prioritize running VMs and, thereby, enforces fairness most effectively. Analytical and simulative investigations show this approach's superiority to existing data center fairness, which is further backed by its practical feasibility being certified by the prototypical extension of OpenStack.

Kurzfassung

Cloud computing ist heutzutage allgegenwärtig, da es erlaubt, Datencenter-Ressourcen fein-granular zu partitionieren und flexibel jedem Gerät, das mit dem Internet verbunden ist, zur Verfügung zu stellen. Diese Partitionierung und Bereitstellung der Ressourcen erfolgt durch Virtuelle Maschinen (VMs), von denen mehrere auf der gleichen physischen Maschine gehosted werden können. VMs werden an Nutzer vermietet und können von diesen wie physische Maschinen benutzt werden. Wegen dieser Möglichkeit, Datencenter-Ressourcen kommerziell, flexibel, effizient und fein-granular zu vermieten, wird Cloud Computing von nahezu allen Firmen und vielen Endnutzern verwendet. Allerdings können kommerzielle Clouds nicht immer Privatheit- und Performanz-Ansprüchen der Kunden gerecht werden oder verstossen gegen Regularien. Daher unterhalten viele Firmen und Institutionen eine private Cloud, die es ermöglicht, die von kommerziellen Clouds gemieteten VMs mit VMs in der privaten Cloud zu komplementieren. Im Gegensatz zu kommerziellen Clouds ist in privaten Clouds die Performanz von VMs nicht durch Service Level Agreements vorgegeben, weshalb VMs in einer privaten Cloud als Prozesse gleicher Priorität behandelt werden. Da Nutzer oft unterschiedlich viele VMs kontrollieren und letztere verschiedene Mengen unterschiedlicher physischer Ressourcen benutzen, führt die Gleichbehandlung von VMs dazu, dass Nutzer unterschiedliche Mengen physischer Ressourcen erhalten.

Ziel dieser Dissertation ist es, durch die Definition eines effizienten Ansatzes, Fairness in privaten Clouds zu gewährleisten: Eine Untersuchung der Schritte, durch die Datencenter-Ressourcen in der Praxis verteilt werden, zeigt, dass Cloud-Ressourcen effektiv kontrolliert werden können, wenn die Prioritäten, mit denen VMs auf die physischen Ressourcen ihres Hosts zugreifen können, angepasst werden. Dies wird nachfolgend als VM-Priorisierung bezeichnet. Zusätzlich wird durch Experimen-

te gezeigt, dass bei der VM-Priorisierung keine Annahmen über Präferenz-Funktionen gemacht werden können. Diese Ergebnisse stehen im Widerspruch zur Mehrheit der Arbeiten über faire Datencenter-Ressourcen-Verteilung, welche versucht, Fairness durch Scheduling zu gewährleisten und hierbei starke Annahmen über Präferenz-Funktionen macht. Da wohldefinierte Präferenz-Funktionen oder wohlstrukturierte Verhandlungsprozesse bei der VM-Priorisierung nicht angenommen werden können, konzeptualisiert diese Dissertation physische Maschinen als „Selbstbedienungs-Buffets" für VMs.

Die Prämisse dieser Dissertation, dass es fair ist, gierige Konsumenten zum Vorteil von weniger gierigen Konsumenten einzuschränken, macht die Definition einer Metrik notwendig, die die Gier von Konsumenten basierend auf den Multi-Ressourcen-Bündeln, welche sie sich aus einem geteilten Ressourcen-Pool genommen haben, quantifiziert. Zu diesem Zweck wird die Greediness- Metrik basierend auf einer Umfrage entwickelt, bei welcher über 600 Teilnehmer zu ihrem intuitiven Gier- und Fairness-Verständnis Auskunft gaben. Diese Greediness- Metrik wird formuliert, evaluiert und für besser als jede allgemein bekannte Metrik zur Quantifizierung von Multi-Ressourcen-Bündeln befunden.

Basierend auf einem neuartigen, formellen Cloud-Model wird die Greediness-Metrik verfeinert, um die Gier von Cloud-Nutzern zu quantifizieren. Die Priorisierung von VMs antiproportional zur Gier ihrer Besitzer wird als Greediness-Fairness definiert. Unter idealistischen Annahmen resultiert die Durchsetzung von Greediness-Fairness in Sharing Incentives (dem Anreiz, Ressourcen zu teilen), Pareto-Effizienz, Strategy Proofness (Nicht-Manipulierbarkeit) und Neid-Freiheit, genauso wie es auch für die Dominant Resource Fairness (DRF) der Fall ist, die anerkannter Maßstab für die faire Datencenter-Multi-Ressourcen-Verteilung ist. Allerdings gibt die Durchsetzung von Greediness-Fairness im Gegensatz zu DRF den Nutzern auch Anreiz (a) ihre VMs korrekt zu konfigurieren, (b) dies gemäß ihrer besten Einschätzung zu tun, falls sie sich unsicher sind, und (c) Arbeitslasten weder auf unnötig viele VMs zu partitionieren, noch die Arbeitslasten künstlich auf wenige monolitische VMs zu konzentrieren. Simulationen zeigen, wie die Durchsetzung von Greediness-Fairness die Ressourcen-Verteilung zwischen physischen Maschinen koordiniert und somit die Fairness zwischen Nutzern verbessert. Die Simulationen zeigen auch, dass die Ressourcen-Verteilung, die aus der Durchsetzung von Greediness-Fairness resultiert, zwischen den

beiden Ressourcen-Verteilungen liegt, welche daraus resultieren, VMs nicht gemäss der Greediness-Metrik zu priorisieren, sondern gemäss der von DRF benutzten Metrik und der naheliegenden Metrik, die alle Ressourcen in einem Bündel aufsummiert. Somit ist die Greediness-Metrik die beste Wahl zur VM-Priorisierung, da hieraus intuitiv faire Ressourcen-Verteilungen resultieren und den Nutzern Anreize gegeben werden, ihre VMs korrekt zu konfigurieren.

Die praktische Anwendbarkeit des von dieser Dissertation vorgeschlagenen Ansatzes wird demonstriert, indem OpenStack durch den prototypischen Nova-Service „Nova-Fairness“ erweitert wird, welcher Greediness-Fairness durchsetzt und mit einer Vielzahl verschiedener VM-Monitore kompatibel ist. Durch Experimente wird der durch die Ausführung von Nova-Fairness verursachte Mehraufwand evaluiert und es wird gezeigt, wie Nova-Fairness die VM-Priorisierung zwischen physischen Maschinen koordiniert, um die Fairness zwischen Nutzern zu erhöhen. Die von Nova-Fairness benötigte Kommunikation zwischen physischen Maschinen ist durch ein hochskalierbares Nachrichtenschema möglich.

Zusammenfassend entwickelt diese Dissertation eine intuitive Definition der Multi-Ressourcen-Verteilungs-Fairness ohne Zugriff auf die Präferenz-Funktionen der Konsumenten zu benötigen. Diese Fairness-Definition wird in Clouds effizient umgesetzt, indem Prioritäten, mit denen VMs auf die physischen Ressourcen ihres Hosts zugreifen können, angepasst werden. Analytische und simulative Untersuchungen zeigen die Überlegenheit dieses Ansatzes gegenüber bereits existierenden Datencenter-Fairness-Ansätzen. Diese Überlegenheit wird zusätzlich durch die praktische Anwendbarkeit untermauert, welche durch eine prototypische OpenStack-Erweiterung demonstriert wird.

Contents

1 Introduction

Cloud computing provides access to a large pool of computational resources to any device that is connected to the Internet. For example, clouds allow executing massive parallelized workloads that take too long, when executed on a single machine, and outsourcing workloads of mobile devices to increase their battery life. While virtualization is at least forty years old [32], it recently was rediscovered to provide data center resources in a highly flexible manner and, thereby, enable cloud computing [49, 91]. Virtualization logically partitions a physical machine, subsequently referred to as *node*, by Virtual Machines (VM), which allows to homogenize the hardware of nodes, isolate users, and efficiently process varying workloads. Therefore, VMs on the same node share the node's resources, such as Central Processing Unit (CPU) time, Random Access Memory (RAM), disk Input/Output (I/O), and network access. Overcommitting nodes allows achieving high resource utilization [7, 22]. Therefore, tradeoffs between the performance of individual VMs sharing a node have to be made, when the resource demands of those VMs peak simultaneously. Performance tradeoffs imply solving a

multi-resource allocation problem, *i.e.*, it is insufficient to allocate resources in isolation [23, 30, 44, 89], as the performance of VMs depends on a combination of resources.

From outside a data center, a VM appears to be a physical machine, *i.e.*, just as users can connect remotely to physical machines inside the data center they can connect to VMs. Thus, VMs allow data center owners, subsequently referred to as Cloud Service Providers (CSP), to efficiently package and lease their computational resources by renting out VMs to private and commercial customers. Therefore, cloud computing has rapidly gained popularity over the last years. A drawback for cloud customers, when renting VMs, is that they have to entrust their data to the CSP and often do not know where their data is located geographically. Furthermore, VMs of different customers share nodes, which may impede the performance of individual VMs or lead to data leakage. Therefore, many companies operate a private cloud to execute performance or privacy critical workloads and, thereby, complement VMs rented from commercial CSPs. Also, institutions such as universities or university chairs operate their own private cloud, *e.g.*, for research purposes or because it provides physical access to the infrastructure. Accordingly, private clouds also gain popularity [16].

In contrast to commercial clouds, VMs of a private cloud are provisioned without payment. This difference is important as a payment is associated with a Service Level Agreement (SLA). In particular, when a VM is rented from a commercial cloud, an SLA prescribes the performance the VM has to deliver. Accordingly, the CSP allocates resources to the VM, such that it does not violate the SLA. In contrast, in private clouds, no SLAs exist and, therefore, VMs are treated as node processes of equal importance. However, the following minimal example shows that this approach is suboptimal. Assume there is only node and two equitable users instantiate one VM and two VMs, respectively. As there is only one node, all three VMs are hosted by this node. All VMs execute a CPU heavy workload, wherefore each of the three VMs attempts to utilize as much CPU time as possible. As VMs are treated as equal processes, all VMs receive an equal amount of CPU time. This implies that the user, who operates two VMs, receives twice as much CPU time as the user, who only operates one VM. As users are equitable, the user, who

only operates one VM, has reason to complain that the other user receives twice the amount of CPU time.

Many institutions manage their cloud resources manually, *i.e.*, individually decide on every request to start a VM and, if need be, manually identify and restrict "misbehaving" VMs during runtime. This process does not scale administratively. For example, even for the minimal example above, adjusting the allocation would take several command line calls (assuming that the unbalanced CPU allocation has already been identified) and the result would be inflexible, *e.g.*, when u_2 shuts down one VM, the allocation has to be corrected again. Therefore, a mechanism is desirable, which minimizes administrative overhead by automatically ensuring certain allocation criteria. Such a mechanism would also allow for introducing novel charging schemes for commercial clouds, such as cloud flat rates [83], where SLAs apply per user and not per VM.

1.1 Cloud Fairness

The two most prominent allocation criteria are efficiency and fairness [10, 12, 13, 15, 18, 30, 44]. While efficiency is important, it can only partially serve as an allocation goal. The reason is that when efficiency, *i.e.*, high usage of resources, is the only criterion, users have the incentive to insert endless loops into their code [30] in order to artificially increase resource utilization and, therefore, efficiency when allocating more resources to them. To avoid that users artificially inflate their resource utilization, it must be ensured that users receive equitable resource shares, *i.e.*, the allocation is fair. As it turns out, fairness is particularly hard to define in clouds, for the following reasons.

Firstly, fairness is an intuitive concept, that is, its definition differs from person to person [83]. While it is generally perceived as fair, to allocate to users equitable (resource) bundles, equitability is hard to define in the multi-resource case, because bundles are not objectively comparable. Cloud users do have different demands over different resources [14, 89, 93, 109]. For example, a certain set of cloud users may require more CPU for their workloads, while others require more RAM. A third user group may deploy the cloud for backups and, therefore, mostly requires disk-space and bandwidth. Hence,

users must receive bundles with different resource ratios, while these bundles cannot be compared objectively and, thus, equitability is hard to measure. In economics this non-comparability of bundles is solved by declaring envy freeness [15] an important fairness criterion, *i.e.*, that no user u prefers to swap the resources u receives with the resources another user receives. Although this defines a sound and formal approach to fairness, its verification requires knowing users' utility functions. A user's *utility function* maps each bundle to a number quantifying the user's valuation for the bundle [95]. In clouds, such utility functions have to be derived from monitoring data, which is not only technically challenging and mathematically complex [114], but may also be hindered by privacy constraints.

Secondly, defining fairness such that it is applicable to clouds is aggravated by the organization of resources: resources are partitioned to different nodes and a user utilizes resources via VMs. A VM is hosted by a node and can only receive resources from this node. Hence, while fairness is to be enforced among users, resources have to be allocated to VMs of which users operate different numbers and the resource demands of VMs (even of the same user) differ.

Thirdly, in clouds users have to configure their VMs with virtual resources, *e.g.*, Virtual CPU (VCPU) or Virtual RAM (VRAM), and these VRs are an indicator of how many resources the VMs will require. Based on this information the CSP places VMs on nodes to optimize resource utilization. Therefore, misconfiguring VMs will unnecessarily decrease the node utilization [80] and it is, therefore, fair to allocate fewer resources to users, who misconfigure their VMs.

1.2 Research Questions

The main goal of this thesis is to develop a mechanism that ensures a fair (and efficient) multi-resource allocation among users of a cloud when the performance of individual VMs is not contracted. Based on the above discussion, therefore, the following research questions guide this thesis.

Research Question 1. How to best control cloud resources? In particular, steps to allocate cloud resources are (a) deciding which VM is started

next, which is most commonly leveraged and termed *scheduling*, (b) deciding which node hosts the VM, which is a combinatorial and not an allocation problem, and (c) allocating node resources to running VMs, whereat time-shared resources are controlled by priorities and not absolute numbers.

Research Question 2. Which general resource dependencies with respect to VM performance exist when allocating node resources to running VMs? If such dependencies exist, they can be modeled as utility functions in the subsequent theoretical investigations; otherwise, very broad and generic assumptions have to be made.

Research Question 3. How can the greediness of consumers be quantified based on the multi-resource bundles they served themselves from a common resource pool? This question arises because Research Question 2 finds that resource and performance dependencies are not predictable when allocating node resources to running VMs. Therefore, cloud fairness cannot be defined via traditional approaches that rely on the existence of well-defined utility functions and a well-structured negotiation process. Instead, this thesis assumes that it is fair to constrain greedy consumers in favor of less greedy consumers when an overload of resources in a shared self-serving resource pool arises and, therefore, requires a sound and intuitive greediness metric.

Research Question 4. How can the definition of greediness be refined to define cloud fairness? This research question addresses the problem that in clouds resources are utilized by VMs, while fairness is to be enforced among users (users have different quotas and operate different numbers of VMs of different configurations). Therefore, the greediness metric has to be refined, such that (a) it is applicable to this complex technical reality and (b) users have the incentive to configure their VMs properly when less greedy users are prioritized at the cost of greedy users. Providing this incentive is advantageous, as the more the VMs' configuration is aligned with their actual consumption, the more efficiently the CSP can place VMs on nodes.

Research Question 5. How can the new cloud fairness definition be practically enforced, when allocating node resources to running VMs?

This problem has to be solved by developing an extension of a publicly deployed cloud software stack, such that the cloud fairness definition is successfully enforced.

1.3 Thesis Contributions

Driven by the explicit research questions posed, this thesis shows that cloud resources are most effectively controlled by changing the allocation of node resources to running VMs and that no assumptions on utility functions can be made during this process. Subsequently, an according comprehensive theoretical model for cloud resource allocation is developed based on which (a) cloud fairness is defined as prioritizing VMs inversely to the greediness of their owners, whereat greediness is determined by a greediness metric that is developed based on a questionnaire among more than 600 participants, and (b) a cloud simulator is developed that compares this greediness metric to other metrics in the context of this fairness definition. Lastly, OpenStack, which is the de facto standard for private cloud software stacks and also serves as a foundation for public clouds, is extended to enforce the proposed fairness definition. Thus, as a result of this approach the outcomes of the thesis cover:

1. Based on (a) an investigation of the practical processes of data center resource allocation and (b) an extensive review of data center fairness approaches, changing the allocation of node resources to running VMs is identified as most potent to control cloud resources. The literature review also identifies characteristics that are highly desirable for data center and, therefore, cloud resource allocation.

2. The performance of VMs executing different workloads is benchmarked to show that VM performance cannot be predicted based on the VM's virtual resources or the resources the VM utilizes. In particular, resource performance dependencies are highly diverse and sometimes even counterintuitive. This proves that assumptions on utility functions, as made by most works on data center resource multi-resource allocation, are inappropriate and, thus, properties proven based on these assumptions hardly hold in reality.

3. A questionnaire among more than 600 participants is conducted and evaluated to conclude on an intuitive understanding of greediness when no utility functions but only the resources consumers served themselves from a shared resource pool are known. The greediness metric [81, 83] is developed as a result and it is shown how it defines an intuitive understanding of multi-resource fairness.

4. A detailed model for cloud multi-resource allocation is developed upon which a very practical and intuitive definition of cloud fairness among users is defined. This fairness definition is tailored to be applied to allocating node resources during VM runtime [80] and incorporates a refined greediness metric. An analytical investigation shows that this fairness definition has all desirable characteristics identified by the literature review and provides incentives for users to configure their VMs correctly [84]. A simulative investigation shows that the refined greediness metric is superior to other metrics, when incorporated into the fairness definition.

5. OpenStack is extended in its implementation by a decentralized fairness service to enforce fairness according to this fairness definition by controlling CPU time, RAM, disk I/O, and network access [80]. The extension's functionality is certified by experiments regarding CPU overhead and fairness enforcement.

1.4 Thesis Outline

Chapter 2 first gives a formal definition of allocation problems followed by a detailed, technical discussion of the organization of data center resources in order to point out where multi-resource allocation problems arise and which steps are necessary to allocate cloud resources. Subsequently, important theoretic aspects of allocation problems are discussed and put into the practical context of data centers. Chapter 2 closes with an exhaustive discussion of approaches to data center multi-resource fairness and compares them with respect to the theoretical and practical aspects discussed beforehand.

Chapter 3 investigates the performance of VMs executing different workloads and shows that the dependency of node resources and their effects on VM performance are highly complex and virtually unpredictable. To account for this lack of utility functions and a well-defined allocation process, fairness is defined as prioritizing consumers inversely to their greediness. The greediness metric is developed based on a questionnaire among more than 600 individuals. Cloud fairness is defined as prioritizing VMs inversely to the greediness of their owners, whereat owner greediness is determined by refining the greediness metric in accordance with a novel model for cloud multi-resource allocation that is developed based on the technical discussion of Chapter 2.

Chapter 4 develops a cloud simulator that prioritizes VMs inversely to their greediness and uses different metrics (including the refined greediness metric) to define greediness. Secondly, an OpenStack extension called nova-fairness is developed to prioritize VMs of cloud users inversely to their greediness.

Chapter 5 shows that the cloud fairness definition proposed in Chapter 3 has the same desirable properties as DRF, which is the most prominent approach to achieve data center fairness. Unlike DRF, this fairness definition also provides an incentive to configure VMs correctly, as shown by analytical and simulative analyses. Next, simulations compare the refined greediness metric to other commonly used metrics and confirm that the refined greediness metric is best suited to determine a prioritization order of VMs. Subsequently, experiments evaluate nova-fairness in terms of CPU overhead and fairness enforcement. Lastly, it is described how the messaging overhead among nodes running nova-fairness can be minimized, making nova-fairness highly scalable.

Finally, Chapter 6 summarizes this thesis' main contributions and highlights its main results. Based on the achieved results conclusions are drawn and an outlook to possible future works presented.

2

Basics and Related Work

This chapter presents formal definitions that are important for the remainder of the thesis and details the technical background of data center resource allocation. Clouds are a special case of data centers, where virtualization is used to provide resources in a highly flexible manner [78].

Section 2.1 formally defines (multi-resource) allocation problems and their important characteristics. Section 2.2 details how data center resources are allocated in practice and where multi-resource allocation problems arise, which identifies how cloud resources are best controlled. In order to investigate which resource dependencies with respect to performance exist in clouds, Section 2.3 compares utility functions that have been proposed to model multi-resource allocation in data centers so far. While Section 2.4 discusses single-resource allocation paradigms that have been extended to data center multi-resource allocation, Section 2.5 describes all key characteristics for practically relevant data center resource allocations. Through this discussion formal requirements, which an intuitive definition of multi-resource fairness must suffice, are identified. Section 2.6 studies and compares in the

closest possible detail approaches that have been suggested to solve multi-resource allocation problems in data centers, thereby providing guidance for how to extend general multi-resource fairness definitions to clouds and how to enforce them. Table 2.1 lists frequently used abbreviations.

2.1 Allocation Problems

Fairness problems arise in many areas of (human) coexistence, *e.g.*, when asking the question of how a governing coalition should be formed in a parliamentary system or how this coalition, once formed, should allocate cabinet ministries [15]. However, (resource) allocation problems are probably the most basic context in which fairness questions arise [78].

Let $R = (r_1, r_2, \ldots, r_\mathfrak{r})$ be a set of $\mathfrak{r}$ resources, where resource $r_i \in R$ is available in the amount of $\overleftrightarrow{r_i}$. An *allocation* of R to $\mathfrak{c}$ consumers in $C = (c_1, c_2, \ldots, c_\mathfrak{c})$ can be denoted by a matrix $A \in \mathbb{R}_{\geq 0}^{\mathfrak{r}\times\mathfrak{c}}$ with $\sum_{j=1}^{\mathfrak{c}} a_{ij} \leq \overleftrightarrow{r_i}$, for all $i \in \{1, 2, \ldots, \mathfrak{r}\}$, where c_j receives amount a_{ij} of r_i. An *allocation problem* is defined as follows:

Given: (a) A set of resources $R = \{r_1, r_2, \ldots, r_\mathfrak{r}\}$, (b) a set of consumers $C = \{c_1, c_2, \ldots, c_\mathfrak{c}\}$, and (c) certain information about the consumers, some of which consumers provide about themselves, or a way to interact with them.

Problem: Define an Allocation Mechanism (AM) that determines an allocation A of R to C with designated characteristics.

Designated characteristics are discussed in Section 2.5. A subset of resources of R is referred to as *bundle*. Accordingly, the resources that $c_i \in C$ receives are referred to as c_i's *bundle* and denoted by A_i. Every consumer c_i is characterized by *utility function* $\mathsf{u}_i : \mathbb{R}^\mathfrak{r} \to \mathbb{R}$, which specifies how happy c_i is with each possible bundle, and $\mathsf{u}_i(A_i)$ is c_i's *utility*. Consumers are assumed to be selfish and to know the AM. Therefore, c_i will provide false information to the AM, if it changes A, such that c_i's utility increases. Knowing the class of consumers' utility functions (cf. Section 2.3), may allow for designing an AM that limits the possibility for strategic manipulation and for proving that it achieves designated characteristics.

Table 2.1: Abbreviations frequently used

Abbreviation	and Term	Defined in Section
AM	Allocation Mechanism	2.1
AP	Allocation Policy	2.1
CEEI	Competitive Equilibrium from Equal Incomes	2.4.2
EF	Envy Freeness/to be envy-free	2.5.1
MR	Microarchitectural Resource	2.2
MRA	Multi-resource Allocation	2.1
NR	Node Resource, *i.e.*, PR or MR	2.2
PE	Pareto Efficiency/to be Pareto-efficient	2.5.3
PR	Physical Resource	2.2
SI	(to provide) Sharing Incentive	2.5.2
SP	Strategyproofness/to be strategyproof	2.5.4
VE	Virtual Encapsulation	2.2
VM	Virtual Machine (special case of VE)	2.2
VR	Virtual Resource	2.2

Allocation problems are classified based on different criteria, for example, whether resources are divisible and/or homogeneous, whether there is only one resource or multiple resources, which information is known about consumers (in the best case all utility functions are known), and the class of utility functions (cf. Section 2.3). An extensive overview of allocation problems classes and their applications can be found in [19].

Many allocation problems allow for the notion that more of a resource is better or at least not worse. This allows to objectively compare the value of two bundles by the size of these bundles, when only a single resource is to be allocated. However, when there is more than one resource, two bundles may contain more of different resources and, therefore, it depends on the consumer, which bundle is more valuable. This aggravates defining fairness, which usually demands to allocate consumers equitable bundles, and finding efficient allocations, as consumers must be allocated resources in different ratios in order to maximize efficiency. As these problems do not exist,

when only a single resource is to be allocated, an important aspect of allocation problems is, whether one or multiple resources are to be allocated. The allocation of multiple resources is referred to as Multi-Resource Allocation (MRA) in this thesis and, accordingly, an allocation problem with multiple resources as MRA problem.

In economics a plethora of allocation problems is investigated [15, 65] and many AMs were proposed. However, these results are not applicable to data centers due to the following three reasons.

1. Economics usually deal with one-time allocation problems, i.e., an allocation problem has to be solved once, while in computer science most resources have to be allocated continuously, which also implies tight runtime requirements. Particularly significant is that reallocating resources in this continuous process may be costly.

2. In data centers many technical constraints prohibit applying AMs developed in economics, as these AMs make idealistic assumptions, *e.g.*, well-defined utility functions and a well-structured negotiation process.

3. Economics usually consider the tradeoff consumers have to make between resources, while in computer science the opposite is the case, as resources are needed in certain ratios [26, 38]. This difference often expresses itself by the class of utility functions that is assumed.

Therefore, basic fairness notions from economics are transferable to data center MRA, while more complex results, such as AMs, cannot. Even many AMs developed for data center MRA make idealistic assumptions, *e.g.*, that resources are always required in static ratios [30, 44]. In order to differentiate theoretic and practical results, an AM for which at least a prototypical implementation for data centers exists is denoted an Allocation Policy (AP). In order to evaluate the practical applicability of AMs, the technical discussion of MRA in data centers follows.

A data center consists of nodes that are connected via middleboxes, such as switches or routers. MRA problems are faced on both devices. Nodes process workloads and communicate with one another and the Internet via flows that traverse middleboxes [78].

On nodes MRA problems arise when allocating Physical Resources (PR) such as CPU time, RAM, disk I/O, and network access and when allocating Microarchitectural Resources (MR), such as cache capacity and memory bandwidth. PRs and MRs are referred to as Node Resources (NR). No approach discussed in Section 2.6 tackles the allocation of PRs *and* MRs. To clearly denote which resources are allocated, abbreviation PRs does not include MRs, although MRs are also physical. On middleboxes MRA problems arise, as flows contend for PRs such as CPU time, RAM, and network access.

Virtualization is applied to homogenize the hardware of nodes, to isolate users, and to efficiently processes varying workloads. Therefore, users process their workloads in Virtual Encapsulations (VE), *e.g.*, Virtual Machines (VM) or containers, such as Docker [25] or LXC [113]. The node that hosts a VE is called the VE's host. Virtualization is applied in most data centers and led to the introduction of the term cloud computing. When VMs are used for virtual encapsulation an entire operating system runs inside each VE. Accordingly, users can handle their VEs, as if they were physical machines, which allows them to deploy almost any software inside and gives the data center operator the flexibility to move VEs between heterogenous nodes. Due to this flexibility from the user's as well as the operator's perspective, renting VMs in a data center is an attractive business model. However, it also implies that users often do not know, where their VMs are geographically located, and data center operators often do not know, which NR utilization to expect from a VM, as almost any software can run inside a VM. VMs and other types of VEs are defined by Virtual Resources (VR), *e.g.*, virtual CPU (VCPU) or virtual RAM (VRAM), which helps to anticipate how much resources they utilize. VRs of a VM are often chosen from a range of *flavors*, *i.e.*, a flavor is a set of VRs that a VM of that flavor has.

As every VM is configured with a set of VRs, VMs not only differ in terms of the NRs they physically utilize but also in terms of their VRs. This makes defining fairness more complex, than defining fairness under the assumption of VEs that are homogenous in terms of their configuration. This thesis focuses on the general case of heterogenous VE configurations and, therefore, defines clouds as data centers, where VMs are used as VEs.

2.2.1 Node Multi-resource Allocation

In order to utilize data center resources users start VEs. Which requests are granted, is decided by the data center operator by *VE scheduling*. VE scheduling is decided based on factors such as how many resources each VE owner currently utilizes by running VEs. The subsequent *VE placement* tackles the problem of finding a suitable node for the VE to be started. VE scheduling and placement is aggravated by potential placement constraints. In particular, the placement constraint of a VE specifies a subset of nodes that qualifies to host the VE. A placement constraint depends on factors such as the nodes' hardware, *e.g.*, the node must have GPUs, software, *e.g.*, the node must run a certain kernel version, or other factors, *e.g.*, the node must have public IP addresses [104]. VE scheduling and VE placement are conducted by the data center's orchestration layer. Optimal VE placement implies solving an APX-hard multi-dimensional BIN-packing problem [59, 112]. As VE placement is a combinatorial and not an allocation problem, it is not subject of this thesis.

After VEs are scheduled and placed, VEs compete for NRs with other VEs on the same node during their runtime and the node's hypervisor (its operating system) is responsible for ruling resulting contention. In other words, the node's hypervisor allocates the node's NRs to running VMs. The allocation of PRs is termed *PR allocation* and the allocation of MRs is termed *MR allocation*.

2.2.1.1 Proportional Priorities

The allocation of time-shared NRs, such as CPU time, disk I/O, and network access, is controlled by Proportional Priorities (PP). In particular, a PP is a non-negative number assigned per NR and VM. The ratios of PPs of VMs sharing a NR define the percentages that VMs receive of this NR. For exam-

ple, when two VMs v_1 and v_2 share a NR r and have PPs 1 and 2, respectively, v_1 receives $\frac{1}{1+2}$ of r and v_2 receives $\frac{2}{1+2}$. In case some VMs do not fully utilize their share, the leftover is allocated in the same manner to VMs that request more of this NR. Thus, PPs do not waste NRs, since a NRs will always be fully allocated, if at least one VM requests it.

The allocation paradigm of PPs is best known as weighted max-min fairness [86]. Operating systems allow to allocate most time-shared NRs by PPs. However, the name to refer to PPs differs depending on the NR: In the context of CPU PPs are termed *shares*, in the context of disk I/O *weights*, and in the context of network access they are associated to certain *queuing disciplines* or *traffic classes*.

2.2.1.2 Dependence of Allocation Steps

The different steps of data center resource allocation (VE scheduling, VE placement, and NR allocation) are intertwined: Depending on which VEs are scheduled and where they are hosted, the contention among VEs for NRs may turn out differently. In reverse, if a node's NRs are highly utilized during runtime, VEs hosted by this node can be live migrated to another node. Furthermore, while the data center orchestration layer can ensure that all users can start VEs of approximately the same amount of VRs, this is of no use for users', if their VEs come off poorly during NR allocation.

2.2.1.3 Job Scheduling

Many works on data center MRA, consider job processing. Here, each user wants to process a job, which is a large batch of data that is partitioned to and processed by tasks. Tasks of the same job/user have homogeneous resource requirements and each task runs inside a VE. Therefore, when considering job processing it is often assumed that all VEs of a user have the same resource requirements. Resource allocation in job processing is controlled by job scheduling, *i.e.*, determining which user can launch a task next. Thus, MRA in job processing is referred to as job scheduling subsequently. However, in general, it cannot be assumed, that all VEs of a user have the same resource requirements, and even for job scheduling this assumptions is often optimistic (cf. Section 2.3.3.2).

2.2.1.4 Comparison of PRs and MRs

MRs, such as CPU cache capacity and memory bandwidth, are inherently substitutable: Data from the RAM is transferred to CPU using memory bandwidth and stored in the CPU cache, such that it does not have to be transferred again and performance increases. Accordingly, when CPU cache size increases, less memory bandwidth is necessary, as more data can be fetched from the cache instead of RAM. Thus, the substitution of MRs [114] does not require additional mechanisms.

Also PRs are substitutable in several cases. For example, (a) paging reduces RAM requirements by use of CPU time and disk space or (b) compression saves storage or bandwidth by use of CPU time [23]. However, paging and compression define higher-level mechanisms, that have to be actively deployed by an operating system to substitute PRs. Furthermore, they may decrease performance. Therefore, contrary to MRs, substitutabilities of PRs cannot be assumed.

2.2.1.5 Overcommitment

Overcommitting nodes allows to achieve a high utilization. For example, VMs are often placed on nodes such that the sum of VRs of a node's VMs exceeds the node's PRs. Overcommit ratios determine the factor by which the sum of VRs of a node's VEs can exceed the node's PRs, *e.g.*, when the CPU overcommit ratio is 15, a node with 4 CPU cores can host VMs with at most 60 VCPUs in total. Accordingly, if a sufficient number of VEs increases the utilization of an overcommitted PR to a certain level, the node's PRs are insufficient to support all VEs' requirements.

Higher overcommit ratios increase the degree of utilization and the chance of overload. Due to this utilization-overload-tradeoff, there is no best overcommit ratio. Thus, overcommit ratios differ among data centers, dependent on the data center operator's risk-adversity and the expected user behaviour. Overcommit ratios also differ among PRs, as the overload of some PRs is more critical than the overload of others. For example, if the node's CPU time is the bottleneck, VEs will run proportionally slower. If RAM is the bottleneck, an over-proportional slowdown occurs (due to heavy pag-

ing). Also, when the VE is a VM it may crash, as a VM's operating system needs a certain minimum amount of RAM. Furthermore, multiplexing CPU is easier, because it is time-shared, while RAM is space-shared. Thus, data center operators select overcommit ratios depending on the PR, *e.g.*, the overcommitment ratio of CPU is often 10 or more, while the overcommitment ratio of RAM is often at most 1.5.

2.2.2 Middlebox Multi-resource Allocation

In addition to general (computing) nodes, middleboxes, such as switches or routers, are the second important instance in data centers, where crucial multi-resource allocation problems arise. Middleboxes connect data center nodes with one another and the Internet. Just as data center users contend for NRs via VEs, they contend for middlebox resources via (network traffic) flows. These flows consist of packets and originate from or terminate at the users' VEs. Middleboxes perform functions as diverse as intrusion detection, HTTP caching, and firewalling. While flows utilize different amounts of bandwidth, depending on their data rate, they also require different amounts of CPU, RAM, and other middlebox resources, depending on which of these functions are applied. Therefore, work on fair middlebox MRA became numerous in the past years [29, 37, 58, 102, 106–108, 116] and is often referred to as fair queueing.

Contrary to NRs, middlebox resources are inherently time-shared and some works on middlebox MRA assume that packets utilize resources successively [29]. This allows defining middlebox fairness over a period of time, not demanding that the resource allocation is fair at every point within that period. In contrast, VEs utilize NRs in parallel and over an undefined time-span. Therefore, fair node MRA demands determining an allocation that is generally fair and not only over a period of time. Furthermore, middlebox fairness definitions can also be based upon packet attributes, such as active/virtual time and timestamps. These definitions are not applicable to node MRA, as VEs do not have these attributes.

As (a) fairness concepts designed for fair node MRA are often extendable to middlebox MRA but not vice versa and because (b) the discussion of mid-

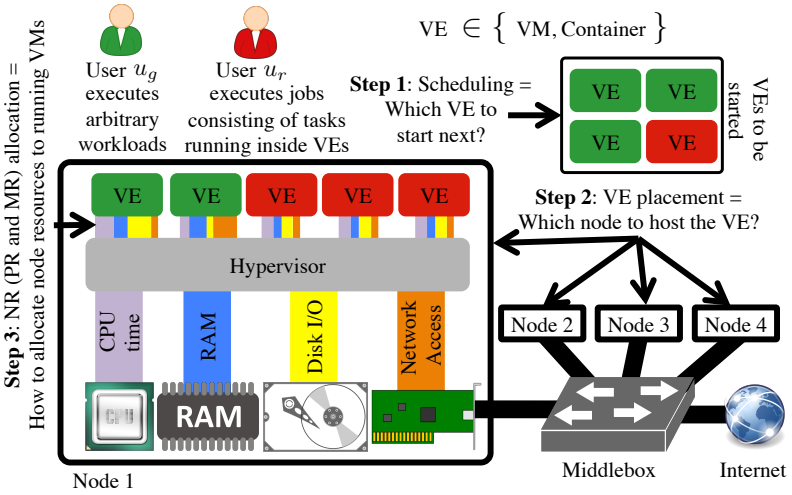


Figure 2.1: MRA in data centers

dlebox MRA demands the introduction of several technical details, this thesis does not discuss the allocation of middlebox resources.

2.2.3 Example

Figure 2.1 illustrates the discussion above by an example. The figure shows a data center that is shared among user u_g and u_r. The data center consists of four nodes that are connected to one another and the Internet via a middlebox.

Node 1 is shown in detail and hosts two VEs of u_g and three VEs of u_r, as depicted by the colors. Both users have requested to start more VEs, wherefore the data center operator has to decide, which of these VEs will be started next and which node will host it. Node 1's hypervisor is responsible for the NR, *i.e.*, PR and MR, allocation to the hosted VEs. As u_r processes tasks of the same job inside the VEs, the NR utilization of u_r's VEs is homogeneous. The according homogeneity of PR utilization is depicted by the same pattern of colored stripes that connect the hypervisor to u_r's VEs. u_g processes different workloads in the VEs, wherefore u_g's VEs utilize different amounts of NRs. As nodes are overcommitted, the requests of the VEs on Node 1 can

exceed the NRs that are available on Node 1. The allocation of MRs to the VEs on Node 1, as well as flows and the middlebox resource allocation are not depicted in the figure.

Figure 2.1 shows that fairness cannot be achieved by leveraging one of the allocation steps exclusively. In particular, if VEs are scheduled such that both users have the same number of VEs, this does not imply fairness. The reason is that VEs can utilize different amounts of NRs even when they have the same VRs, wherefore users utilize very different NR amounts of the data center. Therefore, it is important to leverage the NR allocation to promote fairness. However, this requires to coordinate hypervisors of different nodes. In particular, Node 1 cannot establish fairness among users solely with its local information, as Node 1 does not know how many NRs VEs of the two users receive on other nodes.

Utility functions that have been suggested to model different steps of data center MRA are presented subsequently and contrasted with the practical reality of data centers.

2.3 Utility Functions

Utility functions are considered an essential characteristic of allocation problems [78] and most data center MRA approaches assume a certain class of utility functions to describe how the combination of different resources influences the performance the user perceives. Thus, utility functions are determined by technical factors. However, for data centers realistic utility functions are often hard or even impossible to obtain. In particular, often VEs do not know their utility function. Consequently, utility functions would have to be compiled from monitoring VEs, which is (a) costly, not only because resource monitoring is practically challenging, but also because deriving utility functions from the profiling data is mathematically complex [114], and (b) may violate privacy constraints of users. Nonetheless, many data center MRA approaches make assumptions on utility functions to ease the definition and discussion of fairness. Those assumed utility functions are discussed subsequently and compared in Table 2.2.

Table 2.2: Comparison of utility functions

Utility Function	Area	Drawbacks
Perfectly Complementary	Mainly theory but also scheduling	No substitutabilities
Homogeneous functions of degree one	Mainly theory but also scheduling and MR allocation	Large/unspecific class, thus, hard to achieve results
Leontief	Scheduling	No substitutabilities, no diminishing returns
Cobb-Douglas	MR allocation	Implies substitutabilities

2.3.1 Perfectly Complementary Functions

A utility function is perfectly complementary, if the lack of a resource can never be compensated with another. Thus, for each utility, which a consumer can achieve, there is exactly one bundle without redundant resources that results in this utility (resources are redundant, if they could be removed from the bundle without decreasing the consumer's utility). Although it is theoretically possible to substitute PRs in certain cases, PRs are usually not substitutable (cf. Section 2.2.1.4). Thus, the utilization of PRs can be assumed to be perfectly complimentary.

2.3.1.1 Area

While perfectly complementary utility functions include Leontief utility functions (cf. Section 2.3.3), which are applied to model job scheduling, perfectly complementary utility functions so far have not found another application case in data centers.

2.3.1.2 Drawbacks

By definition perfectly complementary utility functions do not allow to capture substitutability, which apply when allocating MRs.

2.3.2 Homogeneous Functions of Degree one

A function $f : \mathbb{R}^n \to \mathbb{R}$ is homogeneous of degree one, if, for $a \in \mathbb{R}$ and $v \in \mathbb{R}^r$, $f(a \cdot v) = a \cdot f(v)$. In other words, a homogenous function of degree one shows a multiplicative scaling behavior.

2.3.2.1 Area

Amongst others, Leontief and Cobb-Douglas functions are homogeneous of degree one, which are used to model job scheduling (cf. Section 2.3.3) and MR allocation (cf. Section 2.3.4).

2.3.2.2 Drawbacks

It is hard to achieve theoretical results, that are attractive to be applied in practice, when homogeneous utility functions of degree one are assumed, since this class of functions is large (the more specific assumptions about utility functions are, the easier positive results can be achieved).

2.3.3 Leontief Functions

Leontief utility functions are the most prominent example of perfectly complementary utility functions in data center resource allocation and model that a consumer c_i requires resources in a specific ratio. Thus, c_i's Leontief utility function is described by a *resource requirement vector* $v_i \in \mathbb{R}^r_{\geq 0}$ that describes the ratio in which c_i requires r resources. The utility c_i derives from bundle $(x_1, x_2, \ldots, x_r)$ is defined by

$$u_i(x_1, x_2, \ldots, x_r) = \min \left\{ \frac{x_1}{v_{i1}}, \frac{x_2}{v_{i2}}, \ldots, \frac{x_r}{v_{ir}} \right\}. \tag{2.1}$$

[54] provides insights on allocation rules under Leontief utility functions independent of their application in data centers. Leontief utility functions can be extended to finite demands, *i.e.,* consumers have a certain maximum amount of resources they request, which is also known as the knee model [23, 24] and subsequently referred to as finite Leontief utility functions. Leontief utility functions can also be extended to indivisible demands, *i.e.,*

consumer utility increases stepwise and not continuously with the amount of received resources.

2.3.3.1 Area

Leontief utility functions are applied to model job scheduling. As all VEs of the same user are assumed to have the same resource requirements, this implies fixed ratios in which a user requires resources. In combination with the assumption that arbitrary small tasks can be launched, Leontief utility functions are justified.

2.3.3.2 Drawbacks

Jobs may have different classes of tasks with different resource requirements [98], *e.g.*, map and reduce tasks. This poses an argument that Leontief utility functions are too simple to model job scheduling.

Leontief utility functions do not capture substitutability or diminishing returns, *i.e.*, that at some point the satisfaction from receiving a certain resource lessens. [114] points out that computing Leontief utility functions from profiling data is potentially NP-hard, as it requires non-convex optimization.

2.3.4 Cobb-Douglas Functions

When consumer c_i has a Cobb-Douglas utility function, a ratio of resources exists that gives c_i a utility most efficiently. The lack of a resource r relative to this optimal ratio can be compensated by allocating more of other resources. However, this compensation gets more and more "expensive" the more of r is lacking and the more unevenly other resources are used for compensation. Just as for Leontief utility functions, the Cobb-Douglas utility function of c_i is defined by a vector $v_i \in \mathbb{R}^r$. The utility c_i derives from the bundle $(x_1, x_2, \ldots, x_r)$ is then defined by

$$u_i(x_1, x_2, \ldots, x_r) = \prod_{j=1}^{r} x_j^{v_{ij}}. \tag{2.2}$$

Since resource quantities are connected multiplicatively, Cobb-Douglas utility functions allow to model substitutabilities and dependencies. In a normalized presentation, numbers in the vector v_i sum up to 1, which allows to compare different valuations of different consumers on the same scale. By choosing exponents ≤ 1, diminishing returns can be modeled. Cobb-Douglas utility functions can be derived from sampling data by classical regression methods [114].

2.3.4.1 Area

Cobb-Douglas utility functions are proposed in [114] to model utility functions when MRs are allocated.

2.3.4.2 Drawbacks

Cobb-Douglas utility functions allow to substitute resources by other resources. However, not all resources are substitutable (cf. Section 2.2.1.4), which prevents Cobb-Douglas utility functions from being applied.

2.3.5 Example

Figure 2.2 shows the Leontief (*lt*) and Cobb-Douglas (*cd*) indifference curves for requirement vector $(0.3, 0.7)$ and utilities 1 and 3. All bundles described by an indifference curve provide the same utility. As the two Leontief indifference curves are rectangular and parallel to the "resource" axes, utility cannot be increased by only receiving more of one resource. In particular, the bundles described by the vertex of the indifference curves are the minimal bundles, *i.e.*, without redundant resources, that provide the utility 1 and 3. In contrast, the Cobb-Douglas indifference curves converge towards the axes, which shows that substitution, although increasingly expensive, is possible. Indifference curve *ind* belongs to a consumer who only cares about the amount of resources but not their ratio.

In addition to those utility functions discussed, single-resource allocation paradigms are theoretical concepts for resource allocation in data centers.

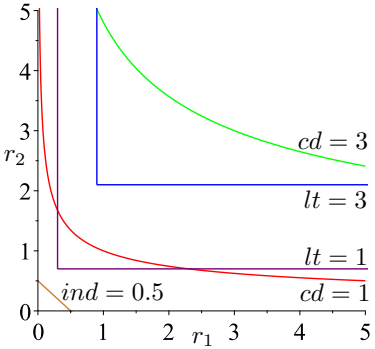


Figure 2.2: Five indifference curves for different utility functions.

2.4 Single-resource Allocation Paradigms

Max-min fairness and proportional fairness are allocation paradigms defined for the allocation of a single resource and extended to MRA [78].

2.4.1 Max-min Fairness

Max-min fairness, also known as the Rawlsian notion of the egalitarian social welfare, is one extreme of fairness notions, as it maximizes the utility of the most unhappy consumer without consideration of the overall efficiency. It is frequently applied in computer science, in particular, bandwidth allocation [27]. [31] extends max-min fairness by placement constraints, to allocate a single resource provided by different nodes.

2.4.2 Proportional Fairness

Proportional Fairness is introduced in [46] and puts a stronger emphasis on efficiency than max-min fairness does. In particular, proportional fairness demands allocating resources to consumers in proportion to how efficiently they use them, *i.e.*, in proportion to how strongly these resources increase their utility. Formally, a proportionally fair allocation A maximizes

$$\sum_{i=1}^{c} \log u_i(A_i). \tag{2.3}$$

To maximize the value that Equation 2.3 yields, more resources have to be allocated to those consumers, whose utility increases fast. However, the log-function implies that the value of the equation increases slower, when allocating resources to a consumer, who already has a high utility, and, therefore, avoids neglecting consumers with slowly increasing utilities.

An alternative definition of a proportionally fair allocation A is that for any other allocation A' the sum of proportional changes to the consumers' satisfaction is not positive, *i.e.*,

$$\sum_{i=1}^{c} \frac{u_i(A'_i) - u_i(A_i)}{u_i(A_i)} \leq 0. \tag{2.4}$$

Proportional fairness can be extended to weights [12]. As noted in [21] the Nash bargaining solution (maximizing the product of utilities) and the Competitive Equilibria from Equal Incomes (CEEI, split resources equally among consumers and subsequently assign prices to the resources, such that trading clears the market) are well regarded solutions in microeconomics for bargaining and fairness and equivalent to proportional fairness for a large class of utility functions, *e.g.*, homogeneous utility functions [65].

2.4.3 Extension to MRA by Bundle Measures

To apply the allocation paradigms above to MRA, every bundle must be characterized by a scalar. In the case of a single resource this scalar is trivially the amount of the only resource in the bundle. However, when multiple resources are allocated, consumers receive a vector of resources. Thus, in order to apply the allocation paradigms to MRA, a bundle measure is necessary, *i.e.*, a metric that objectively (and not on a per consumer basis) quantifies the value of bundles and, thereby, maps allocation vectors to scalars.

The most intuitive bundle measure is the L_1 norm, which determines the value of a bundle by adding up how much of the different resources relative to the overall supply the bundle contains. Asset fairness [30] extends max-min fairness to MRA by using the L_1 norm as bundle measure. The L_∞ norm defines the value of a bundle by the bundle's dominant share, which is the largest amount relative to the overall supply of any resource in the bundle,

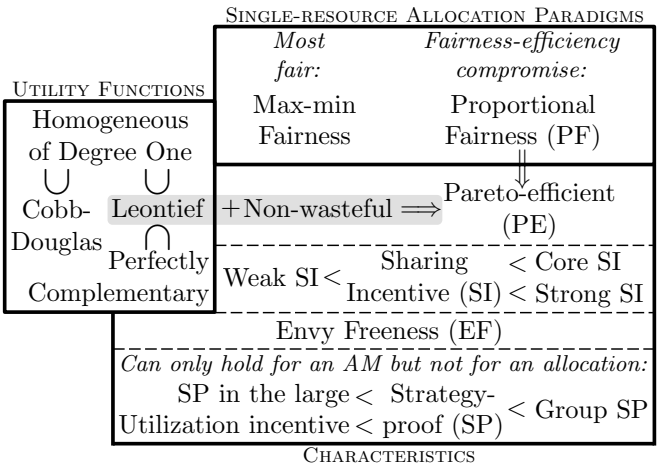


Figure 2.3: Relation of utility functions, single-resource allocation paradigms, and allocation characteristics

i.e., only the resource that is contained most of in the bundle is relevant for the bundle's value. Dominant resource fairness (cf Section 2.6.1) extends max-min fairness to MRA by using the L_∞ norm as bundle measure. Bundle measures are not constrained to norms but every monotonic and continuous function can be used as bundle measure.

Extending max-min fairness to MRA by using bundle measure M is abbreviated by M-max-min fairness in this thesis. Therefore, asset fairness is a synonym for L_1-max-min fairness and dominant resource fairness is a synonym for L_∞-max-min fairness.

When single-resource allocation paradigms are extended to MRA, they ideally result in the following characteristics.

2.5 Characteristics

The following fairness and efficiency characteristics recurring throughout literature are considered relevant for data centers [78]. The relations of these characteristics and the definitions of previous Sections 2.3 and 2.4 are illustrated in Figure 2.3. The figure depicts the inclusion of utility functions by the rotated $\subset$ symbol, $<$ denotes that the characteristic on the right is stronger

than the characteristic on the left, and $\Longrightarrow$ indicates which definitions imply Pareto efficiency.

2.5.1 Envy Freeness (EF)

Envy freeness ensures that no consumer can complain that another consumer receives more. An allocation is envy-free (EF), if no consumer c prefers the bundle of another consumer over the bundle c receives [13, 23, 30]. EF is essential for an intuitive understanding of fairness, since envy is a clear sign of unfairness. Its verification and enforcement requires to know consumers' utility functions, as the latter are necessary to determine, whether consumers do envy other consumers.

If the $\mathfrak{c}$ consumers arrive in $\mathfrak{c}$ steps and never release resources once allocated, an allocation is dynamic EF, if a consumer c_i envies a consumer c_j only if c_j arrived before c_i did and c_j has not received resources since c_i arrived [45].

An AM is (dynamic) EF, if it always returns an allocation that is (dynamic) EF.

2.5.2 Sharing Incentives (SI)

Sharing Incentives (SI) ensures that no consumer is better off, when resources are shared naively. The equal division divides every resource equally among consumers (or, if consumers are associated with weights, proportionally to the consumers' weight). An allocation provides SI, or, for the sake of brevity, an allocation is SI, if every consumer c is at least as good off as c would be under the equal division.

SI finds notably more reference in the context of data centers than in economics and is widely accepted as a desirable goal for data center Multi-Resource Allocation (MRA). Thus it surprises that SI is ambiguous, when the realistic assumption is made, that resources are partitioned to different nodes. Therefore, [103, 109] refine SI as follows. An allocation A is strong SI, if every consumer c is at least as good off under A, as c would be when the resources of every node are divided equally. A is weak SI, if an equal division exists, that gives every consumer at most as much utility as A. An equal division is an allocation that gives each consumer a $\mathfrak{c}$-th of every resource, whereat it is not specified how resources are provided by individual nodes.

Therefore, usually equal divisions exist that allocate consumers zero of some resource on every node. This makes weak SI indeed a weak characteristic.

Coalitional game theory considers the question of how a coalition of agents should divide the coalition's payoff. In this context, coalitional game theory defines the "core of payoff divisions" as the set of payoff divisions, for which no agents have incentive to leave the coalition to form a smaller coalition that allows them to receive a bigger payoff [95]. This motivates to define that an allocation is *core SI*, if no subset of consumers can do better by forming a coalition to divide their equal shares exclusively in this coalition.

An AM is SI, strong SI, weak SI, or core SI, if it always returns an allocation that is SI, strong SI, weak SI, or core SI, respectively.

2.5.3 Pareto Efficiency (PE)

Pareto efficiency ensures that no allocation exists that is unambiguously more efficient than the current allocation. Pareto efficiency does not capture fairness but efficiency and is desirable under virtually all circumstances. An allocation is Pareto-efficient (PE), if no allocation exists that makes at least one consumer better off, while making no consumer worse off [38]. PE is necessary for overall efficiency but not sufficient.

[38] introduces the concept of *non-wastefulness*, which demands that no resource desired by any consumer is left unallocated and that no consumer c receives resources c has no use for. It is pointed out that under perfectly complementary utility functions any non-wasteful allocation is PE. This trivial achievement of PE increases the relevance of fairness.

Proportional fairness implies PE: Assume proportionally fair allocation A is not PE. Because A is not PE, an allocation A' exists, that gives each consumer at least the same utility and at least one consumer more utility. Then the left side of Equation 2.4 is positive. However, then A is not PE.

If the $\mathfrak{c}$ consumers arrive in $\mathfrak{c}$ steps and never release resources once allocated, achieving PE and EF in every step is impossible: PE demands to allocate most resources to the consumers who have already arrived, which are then envied by consumers arriving in subsequent steps. Therefore, an allocation A^k in step k is defined to be dynamic PE, if A^k allocates at most a $(k/\mathfrak{c})$-fraction of every resource and is PE among those allocations that allocate at

most a (k/c)-fraction of every resource, *i.e.*, none of these allocations makes at least one consumer better off, while making no user worse off. While dynamic PE and EF are generally incompatible, dynamic PE and dynamic EF are not [45]. An allocation is defined to be cautious dynamic PE if it can ultimately guarantee EF and is PE among those allocations that can ultimately guarantee EF.

An AM is (dynamic/cautious dynamic) PE, if it always returns an allocation that is (dynamic/cautious dynamic) PE.

2.5.4 Strategyproofness (SP)

Contrary to former characteristics, strategyproofness can only hold for an AM but not for an allocation. An AM is strategyproof (SP) or truthful, if consumers cannot increase their utility by providing false information, *e.g.*, the AM gives incentives to consumers to report their true utility function. SP is generally hard to prove and often impossible to achieve.

Group SP is a stronger version of SP and demands that, whenever a coalition of consumers coordinates their misreporting, there is at least one consumer in the coalition that will be worse of. This straightforward extension of SP to coalitions is similar to the extension of SI to coalitions in Section 2.5.2.

In practice SP is not necessarily as important as one might expect: [115] points out "that as long as fairness (however one may decide to define it) is maintained, there is no way that the system can distinguish between a client that genuinely requires more resources and one that is trying to game the system." [12, 13] shows that APs exist that are not SP in theory, but consumers cannot game the AP in practice, because they lack necessary information. [114] shows that APs exist that are not SP in general but for a sufficiently large number of consumers. This characteristic is termed SP in the large.

Utilization incentive is introduced in [77] and a special-case of SP, as it demands that no consumer can benefit from under-reporting demands. Utilization incentive ensures that consumers do not utilize less of an underutilized resource to receive more of a resource under contention.

Ideally the theoretical concepts discussed so far are achieved by data center MRA approaches.

2.6 MRA Approaches for Data Centers

While basic and generic notions for fairness and efficiency have been discussed above, MRA fairness approaches for data centers need to be detailed [78]. Thus, Table 2.3 summarizes the respective findings and this section discusses existing approaches in proportion to papers published on them. Thus, shortcomings of approaches are discussed in the same set of proportions. The abbreviations in the column *Approach* define in a single row of this table one separate approach, which is detailed in the subsections below. Each subsection discusses first for which area of data center resource allocation this approach is designated, which also motivates utility functions (summarized in Table 2.3 columns *Area* and *Utility Functions*). Secondly, each subsection discusses the fairness definition proposed by the approach (*Fairness Definition* column). Whether an approach is SP, SI, PE, or EF is specified in the four rightmost columns, while not explicitly stated in the sections. Furthermore, the discussion here details the approaches' implementations. Finally, if applicable, shortcomings and extensions are discussed.

2.6.1 Dominant Resource Fairness (DRF)

Dominant Resource Fairness (DRF) [30] is the most prominent approach to MRA fairness in data centers. DRF has been extended in many directions and compares bundles by their dominant resource, *i.e.*, the value of a bundle is defined by the amount (normalized by the overall supply) of the resource that is contained most in the bundle. Therefore, all other resources in the bundle are ignored.

2.6.1.1 Area and Utilities

DRF is applied to job scheduling, wherefore Leontief utility functions are assumed (cf. Section 2.3.3.1).

2.6.1.2 Fairness Definition

DRF defines fairness as dominant resource fairness (cf. Section 2.4.3). Thus, every user receives the same dominant share, if users request a strictly positive amount of every resource. Due to the assumption of Leontief utility

Table 2.3: Comparison of approaches

Approach	Fairness Definition	Area	Utility Function	SP	SI	PE	EF
BBF	All users receive at least equal share on one bottleneck resource	PR allocation	Perfectly complementary	No	Yes	Yes	No
DRF	Max-min fairness + L_∞ bundle measure	Scheduling	Leontief	Yes	Yes	Yes	Yes
GRF	Max-min fairness + monotonic and continuous bundle measure	Theoretical insights	Perfectly complementary	-	Yes	Yes	-
REF	Determine allocation that is Nash bargaining solution and CEEI	Micro architect	Cobb-Douglas	Yes	Yes	Yes	Yes
PDF	Proportional fairness + L_∞ bundle measure	Scheduling	Leontief	No	Yes	Yes	Yes
PFT	Proportional fairness	Theoretical insights	Homo. degree 1	Yes	-	No	-
RRF	Loosely defined by algorithm	PR allocation	Not needed	Yes	Yes	Yes	-
PBS	Suggestions for priority functions	Scheduling	Not needed	-	-	-	-
TSF	Max-min fairness + Task Share bundle measure	Scheduling	Leontief with constraints	Yes	Yes	Yes	Yes
G^δ	Max-min fairness + GM bundle measure	PR allocation	Not needed	Yes	Yes	Yes	Yes

functions, the dominant share that a user receives precisely determines the amount of all other resources received.

2.6.1.3 Implementation

A DRF AM is proposed in [30], which allows users to start tasks with different resource requirements. A DRF AP is implemented in the Mesos cluster resource manager to show that DRF improves throughput and fairness compared to standard slot-based fair sharing schemes [30, 40].

2.6.1.4 Shortcomings

While DRF is the de facto standard for MRA in data centers, it shows shortcomings.

2.6.1.4.1 Assumptions and Practice

All proofs for DRF's desirable characteristics are based on Leontief utility functions. However, the DRF AP proposed in [30] allows users to change the ratio of their resource requests and users may indeed have more complex utility functions (cf. Section 2.3.3.2). [60] provides an example of an unfair DRF allocation, when Leontief utility functions do not hold.

2.6.1.4.2 Intuitive Fairness

DRF chooses the L_∞ norm to quantify bundles, but never justifies this choice. [38] points out that many other functions including all $L_{i \in \mathbb{N}^+}$ norms are feasible. Each alternative results in a unique allocation that can be found in polynomial time. It is hard to argue in favor for one of these alternatives. Moreover, the L_∞ norm is inappropriate, when non-perfectly complementary utility functions are assumed, because the L_∞ norm is oblivious to all non-dominant resources in a bundle. However, even for Leontief utility functions DRF allocations can be intuitively unfair: Consider the following example, which points to a problem similar to the problems in [11, Section 3.2], [56, Section 7], and Section 3.2.1. Assume there are two resources, each of which is available in one infinitely divisible unit, and three users with demands $(1, 0)$, $(0, 1)$, $(1, 1)$, *i.e.*, the first user only requests the first resource, the second user requests only the second resource, and the third user requests

both resources in equal amounts. DRF allocates the first user 0.5 units of the first resource, the same for the second user and second resource, and the third user receives 0.5 units of both resources. Therefore, the third user receives the same amount of resources as the other two users combined, which is intuitively unfair (cf. Section 3.2.1). This example can be extended to $m > 2$ resources and $m + 1$ users, where one user receives the same amount of resources as all other m users combined. In these examples, the DRF AM gives incentive to users with asymmetric utility functions to form a coalition with their equal shares and divide it among them. Therefore, this example shows that, while DRF is SI, it is not core SI (cf. Section 2.5.2). Notably, even if the newly formed coalition allocates their resource share according to DRF, the coalition partners are better off than in the "grand coalition". Therefore, even without new allocation strategies, users can increase their allocation simply by forming coalitions.

2.6.1.4.3 Efficiency

[74] justifies DRF's potentially poor overall efficiency by impossibility results for AMs that are SP or EF or SI. However, [12, 13] show that applying proportional fairness to the dominant shares (instead of the max-min fairness) greatly increases overall efficiency (cf. Section 2.6.5). Although applying proportional instead of max-min fairness makes the AM non SP, [12, 13] argue that in data centers users nonetheless have insufficient information to game the AM.

2.6.1.5 Extensions

[30] left many questions open that are theoretically interesting and made many assumptions that are practically too simplistic. These theoretical questions were investigated in follow up works just as DRF was extended to the more realistic assumptions that user demands are finite and resources are distributed over multiple nodes. Furthermore, DRF was extended from resource sharing in space to resource sharing in time, to make it applicable to middleboxes, which is not subject of this thesis, and it was extended by user hierarchies. These extensions are discussed next and constitute important research directions for less developed approaches.

2.6.1.5.1 Theoretical Extensions

[74] investigates DRF from an economic point of view and establishes the groundwork for many other DRF extensions. In particular, [74] shows that DRF retains its desirable characteristics when users are weighted and even when they do not require every resource (the latter assumption was crucial for many arguments in [30]). Also, DRF's poor performance with respect to overall efficiency is justified by impossibility results for AMs that are SP, or EF or SI. It is shown that perfect fairness cannot always be achieved, when users have indivisible demands. Additionally, an AM that achieves fairness to the best possible extent given these impossibilities is presented. It is argued that this algorithm can be implemented for dynamic scenarios, *i.e.*, where users change their demands over time.

A different economic perspective on DRF can be found in [38], which extends DRF in terms of utility functions and bundle measures to define Generalized Resource Fairness (cf. Section 2.6.3).

2.6.1.5.2 Finite Demands and Joining of Users

[45] presents (a) a DRF AM that is SI, dynamic EF, dynamic PE, and SP in every step and (b) a DRF AM that is SI, EF, cautious dynamic PE, and SP in every step. [55] improves the runtime of the first AM introduced in [45] from pseudo-polynomial to linear. The comparison to an optimal offline AM shows that the AM, although online, is nearly optimal. [57, 62] present a DRF AM that (a) works for finite Leontief utility functions, (b) is SI, dynamic EF, dynamic PE, and SP in every step, and (c) runs in $\mathcal{O}(u^2 \log u)$, where u is the number of users. This runtime is an advantage over the (also) polynomial AMs of [45].

[56] extends DRF to finite Leontief utility functions by comparing allocations by their lexicographic value. In particular, the fairness of allocations is compared by ordering the dominant shares increasingly and comparing the resulting vectors lexicographically. An AM is presented that determines an allocation that is optimal according to this metric. It is proven that the AM is group SP, EF, PE, and SI. Although the definitions and AM work for all norms, *i.e.*, $L_{i \in \mathbb{N}^+ \cup \infty}$, arguments in favour of using the L_∞ norm are presented.

[28, 61, 85, 94, 103, 109, 117] extend DRF to the case, where resources are distributed over multiple heterogenous nodes. [28, 61, 85] assume indivisible demands, while [94, 103, 109, 117] assume infinitely divisible demands. All approaches, except [94], define the dominant share of a user relative to the sum of node resources.

[94] is the first work to extend DRF to multiple nodes. A formula of [44] is extended to define the Optimization Problem (OP). This extension defines the value user u receives as the sum of u's dominant shares over all nodes. As these dominant shares are calculated relative to the respective node's capacity, the same dominant share may correspond to different amounts of resources depending on the node. However, this is not addressed. An AM for solving the OP repeatedly to converge toward a solution is presented and the authors demonstrate fast convergence. However, the number of variables in the OP is equal to the product of nodes and users. Thus, the question of how long one iteration takes arises but is not addressed.

[85] explicitly addresses VEs scheduling *and* VE placement. [85] focuses on PE instead of EF or SP. This focus is justified by impossibility results in [74] for indivisible demands. No formal proof of PE is provided.

[28] proves that no deterministic AM is SI, PE, and SP simultaneously. This result motivates the subsequently presented non-deterministic AM that is ex-ante (a) dominant resource fair, (b) PE, and (c) SP under the idealistic assumption that all users are utility maximizers, which implies that they must not be risk averse. The concrete algorithmic implementation of this AM is left open.

[109] is an extended version of [103]. [103, 109] formulate an OP with as many variables as the product of nodes and users (just as [94]) and discuss the ambiguity of SI, when resources are distributed over multiple nodes. The OP results in a DRF allocation that is EF, PE, group SP, weak SI, and significantly better than enforcing DRF at each node individually. It is argued that not achieving strong SI is the tradeoff for the high overall efficiency. It is also sketched how this approach can be extended to (a) weighted users, (b) finite demands, and (c) indivisible demands.

[117] highlights the complexity of finding a solution for the OP formulated in [103, 109]. Thus, [117] presents two distributed AMs to approximate DRF in a distributed environment. It is assumed that every node can only communicate with few other nodes about known users and their bundles. Simulations show that these distributed AMs scale and achieve a similar level of DRF as a centralized implementation.

2.6.1.5.4 User Hierarchies

[11] extends DRF by user hierarchies. The presented AP is implemented for Hadoop YARN and allows to organize users in a weighted tree structure, where the leaves are users and each inner node has a positive weight, which denotes its relative importance. [11] points out a key-feature called sibling sharing, which enables resources to stay within a sub-organization in the hierarchy, when users finish within that sub-organization.

2.6.2 Bottleneck-based Fairness (BBF)

Bottleneck-based fairness (BBF) is introduced in [24] for virtualized environments and demands that every user, if not satisfied, receives a "fair" share of at least one resource that is depleted.

2.6.2.1 Area and Utilities

BBF is designated to be enforced by hypervisors among VMs to allocate a node's PRs. Most work on BBF assumes finite Leontief utility functions [23, 24], while BBF is also applicable to perfectly complementary, continuous, and strictly monotonic utility functions [38].

2.6.2.2 Fairness Definition

An allocation is bottleneck-based fair or allows for "no justified complaints", if every user either receives all requested resources or at least the equal share on a bottleneck resource, *i.e.*, a resource which is fully utilized, and the other resources in proportion to this bottleneck resource based on the user's utility function.

2.6.2.3 Implementation

[38] presents a polynomial time AM to find a BBF allocation for finite Leontief utility functions. While the AM is of theoretical interest, it is too complex to be applied in practice.

[115] describes the following multi-resource on-line scheduling policy that achieves BBF without knowing users' utility functions in advance. For a resource r and a user u, u's gap of r is the difference between what u receives of r and the equal share of r, *i.e.*, the less u receives of r, the larger u's gap of r. u's minimal gap over all bottleneck resources is u's priority that is used to schedule all resources. A problem with this approach is that bottleneck resources are often determined by allocation decisions taken before.

2.6.2.4 Shortcomings

Enforcing BBF is neither SP nor EF. However, [13] shows that in realistic settings users do not have enough information to successfully game a BBF AM.

2.6.2.5 Extensions

[23] proves that for finite Leontief utility functions a BBF allocation always exists, even if there are multiple bottleneck resources. [38] proves that for all perfectly complementary, continuous, and strictly monotonic utility functions a BBF allocation always exists.

[13] defines *Bottleneck Max Fair (BMF)* allocations as those BBF allocations, where every user receives a bundle of at least one bottleneck resource that is maximal for that resource. It is demonstrated that a BMF allocation (a) always exists, (b) is similar to the proportionally fair allocation (cf. Section 2.6.5) but simpler to calculate, and (c) is more efficient than the DRF allocation.

2.6.3 Generalized Resource Fairness (GRF)

Generalized Resource Fairness (GRF) is defined in [38] and puts DRF in a theoretical context by allowing more utility functions and bundle measures. As the only constant commonality among DRF and GRF is max-min fairness,

GRF can also be viewed as an investigation of max-min fairness in the context of multiple resources.

2.6.3.1 Area and Utilities

No practical application case is proposed, as GRF is developed out of technical curiosity. However, GRF theoretically has several application cases, as perfectly complementary utility functions are assumed.

2.6.3.2 Fairness Definition

Generalized Resource Fairness (GRF) generalizes DRF with respect to the bundle measure and utility functions [38]. In particular, GRF allows all monotonic and continuous functions as bundle measures and all perfectly complementary utility functions. Therefore, DRF is a special case of GRF and the only constant commonality of DRF and GRF is the enforcement of max-min fairness. Another special case of GRF is asset fairness. [38] shows that for every instantiation of GRF the according allocation is unique and can be found in polynomial time. GRF is PE, but [38] does not discuss characteristics such as SP, SI, or EF.

2.6.3.3 Implementation

[38] presents a pseudo-polynomial AM to calculate a GRF allocation. The AM needs access to users' utility functions. [56] presents an AM to find allocations for a constrained version of GRF, when resources are distributed over different nodes (cf. Section 2.6.1.5.3). This version of GRF is constrained to finite Leontief utility functions and norms as a bundle measure.

2.6.4 Resource Elasticity Fairness (REF)

Resource Elasticity Fairness (REF) is defined in [114], complies with well regarded fairness definitions from economics, and, therefore, has many desirable characteristics.

2.6.4.1 Area and Utilities

[114] aims to allocate cache capacity and memory bandwidth and profiles different applications to convincingly argue that Cobb-Douglas utility func-

tions are well suitable to model diminishing returns and substitution effects of these MRs. It is argued, that also the dependency of other MRs, *e.g.*, number of processor cores, can be modeled by this class of utility functions.

2.6.4.2 Fairness Definition

Resource Elasticity Fairness (REF) is defined via an AM. The produced solution is shown to be a Nash bargaining solution and a CEEI, which are both well regarded fairness definitions from economics.

2.6.4.3 Implementation

The proposed AM receives user's reported utility functions as input and is SP in the large already for tens of users. Cycle-accurate processor and memory simulators are used to show that the AM causes an overall efficiency loss of less than 10% (efficiency is measured by the number of instructions committed per cycle), compared to when no fairness mechanism is active.

2.6.5 Proportional Dominant resource Fairness (PDF)

Proportional Dominant resource Fairness (PDF) is introduced in [12, 13] and extends proportional fairness to MRA by using the L_∞ norm as bundle measure. Therefore, PDF can be viewed as a modification of DRF by replacing the max-min-fairness with proportional fairness.

2.6.5.1 Area and Utilities

PDF is designated for job scheduling, wherefore Leontief utility functions are assumed. (cf. Section 2.3.3.1).

2.6.5.2 Fairness Definition

PDF extends proportional fairness to MRA by using the L_∞ norm as bundle measure. As proportional fairness trades off a certain degree of fairness for efficiency, PDF shows a higher overall efficiency than DRF.

2.6.5.3 Implementation

[13] presents AMs to implement the PDF for scheduling as well as queuing problems.

2.6.5.4 Shortcomings

PDF uses Leontief utility functions to model job scheduling. However, Leontief utility functions are often too simple for this purpose (cf. Section 2.3.3.2).

PDF is not SP. However, [12, 13] argues that under the realistic assumption that the population of users follows a stochastic process, users are unable to game PDF, even if they have partial knowledge about the requirements of other users.

2.6.6 Proportional Fairness Theory (PFT)

Proportional Fairness Theory (PFT) investigates proportional fairness from an mechanism design perspective. The results achieved in [21] are theoretically interesting but practically not applicable due to very low efficiency.

2.6.6.1 Area and Utilities

[21] aims at theoretical insights and considers homogeneous functions of degree one as utility functions. The results achieved in [21] are theoretically applicable to the allocation of MRs and job scheduling, because Cobb-Douglas functions and Leontief functions are homogeneous of degree one. However, for these special cases, stronger results exist (cf. Section 2.6.4 and 2.6.5).

2.6.6.2 Fairness Definition

PFT defines fairness as proportional fairness. SP is achieved by largely sacrificing efficiency.

2.6.6.3 Implementation

[21] presents an AM to which each user announces a bundle of resources. The AM then allocates the announced bundle scaled down by a factor, which depends on the bundle's comparativeness. The AM is SI and gives each user c at least $1/e = 0.368$ of c's proportional fair utility, *i.e.*, of the utility c would get from a perfectly proportionally fair allocation.

2.6.6.4 Shortcomings

In order to be SP, the AM may hold back resources. This approach is theoretically justified by showing that no AM that is SI can guarantee to every user a utility greater than 0.5 of the proportional fair utility. Although the efficiency of the AM is low, the AM is interesting, because it gives a constant factor approximation to every user of the user's proportionally fair allocation.

2.6.7 Priority-based Sharing (PBS)

[47, 48] introduce fairness via Priority-based Sharing (PBS) and point out that clusters, which provide several PRs such as CPU, RAM, GPUs, and HDD storage, are often shared based on priorities, which are calculated solely based on utilized CPU time. Accordingly, situations are possible, where a user utilizes 100% of a machine's RAM, but only 10% CPU, which makes the remaining 90% of CPU unusable, while the user is only accounted 10% usage.

2.6.7.1 Area and Utilities

PBS is applied to job scheduling and makes scheduling decisions based on the current resource utilization of users, wherefore utility functions are not specified.

2.6.7.2 Fairness Definition

[47, 48] introduce fairness via penalty functions. A penalty function aggregates a user's current multi-resource utilization to a (priority) scalar. The user with the highest priority can launch the next task.

2.6.7.3 Implementation

An AP implemented for the TORQUE resource manager is used in production as of Nov. 2013 [47].

2.6.7.4 Shortcomings

As utility functions are not specified, characteristics SP, SI, PE, and EF cannot be discussed.

2.6.7.5 Extensions

[47, 48] identify the following penalty functions as most advantageous.

2.6.7.5.1 Arithmetic Functions

[48] is dedicated to finding a penalty function P that suffices the following five requirements.

1. P takes several resources into account.
2. P calculates the penalty based on several resources, not only based on the dominant share: When users have the same dominant share, their priority is distinguished by the utilization of other resources.
3. P has to result in a maximal penalty, whenever a user completely utilizes a resource of a node. In particular, this maximal penalty has to hold irrespective of how much is utilized of the other resources, since it blocks the respective node.
4. P increases linearly with the utilization of resources, to not give incentive to artificially partition workloads.
5. P shall allow for associating weights with resources.

Different penalty functions are compared with respect to these requirements and it is proven that Requirement 2, 3, and 4 cannot be achieved simultaneously. It is shown that defining P as the sum or product of (relative) utilizations yields penalty functions that suffice all but Requirement 3 or 4, respectively.

2.6.7.5.2 Improved Processor Equivalent

[47] improves the Processor Equivalent (PEQ) metric. The improved PEQ is available in the Maui and Moab schedulers [1]. A task's PEQ is the dominant share over CPU and RAM relative to the data center-wide capacity times the data center-wide CPU capacity.

A drawback of PEQ is that it does not account for, when a node is completely blocked: consider node n that has more RAM than CPU relative to

the data center's average. Furthermore, assume n is completely occupied by task t_i, because t_i utilizes all of n's RAM. Then, t_i's PEQ is determined by its RAM utilization. Now assume, instead of t_i, task t_j runs on n and uses all of n's CPU. t_j's utilization of RAM is sufficiently small, such that t_j's PEQ is determined by its CPU utilization. Accordingly, t_j's PEQ smaller than t_i's PEQ. However, because both t_i and t_j entirely utilize one of n's resources, both block n completely. Therefore, t_i and t_j effectively occupy the same data center resources and should have the same PEQ. To overcome this problem, [47] calculates a task's penalty as the dominant share with respect to the node capacity instead of data center-wide capacity. To make dominant shares comparable across nodes they are normalized by the nodes' CPU capacity (furthermore, wall time normalization is applied). While this determines an improvement, a task's penalty is still sensitive to scheduler decisions. That is, depending on a task's host, the task's PEQ is different. This is undesirable, because tasks can only take limited influence on the scheduler. To achieve scheduler insensitivity, the penalty of a task is calculated with respect to each node type that could host the task. The smallest of these penalties defines the task's final penalty. The penalty of a user is the sum of penalties of the user's tasks.

2.6.8 Reciprocal Resource Fairness (RRF)

Reciprocal Resource Fairness (RRF) is similar to asset fairness and loosely defined in [60] by an AP.

2.6.8.1 Area and Utilities

RRF is designated to share PRs among VMs that belong to users, who have rented these PRs together. The sharing is established based on ad hoc needs, wherefore utility functions are not specified.

2.6.8.2 Fairness Definition

RRF generalizes max-min fairness to multiple resource types and is designed to be (a) SI, (b) SP, and (c) to provide gain-as-you-contribute fairness. Goal c means that a user receives spare PRs from other users proportionally to how

many PRs this user cedes to other users. While a formal definition for Goal c is not available, [60] states that asset fairness is considered optimal.

2.6.8.3 Implementation

A prototypical RRF AP for Xen is provided. Firstly, PRs are traded between users based on their needs. Secondly, VM priorities are adapted by the users, who own the VMs, depending on the VMs' requirements.

2.6.8.4 Shortcomings

When a VM does not utilize all PRs it is entitled to based on its "shares", those PRs are not always fully utilized by other VMs but RRF counts those PRs, as if they were. As Goal c and the AP are not uniquely formalized, RRF cannot be considered a concrete definition of fairness but only as an AP.

2.6.9 Task Share Fairness (TSF)

Task Share Fairness (TSF) is defined in [104, 105] and based on max-min fairness. TSF allows for placement constraints and has many desirable characteristics.

2.6.9.1 Area and Utilities

TSF is applied to job scheduling, wherefore Leontief utility functions are assumed (cf. Section 2.3.3.1). However, each user also states a boolean placement constraint vector that indicates for each node in the data center, if the node can host the user's tasks. Furthermore, it is assumed that resources are partitioned to different nodes.

2.6.9.2 Fairness Definition

Fairness is defined as max-min fairness among task shares of users. The task share of a user u is defined as the ratio of the number of tasks u currently runs and the number of tasks u could run, if u had no placement constraints and there were no other users in the data center.

[104] shows that TSF is SI, SP, EF, and PO. It is highlighted, that under placement constraints, no extension of DRF to multiple nodes (cf. Section 2.6.1.5.3) has these characteristics.

2.6.9.3 Implementation

A TSF AP for Apache Mesos allows the user with the lowest task share to launch the next task. Placement constraints are specified by white- or blacklists.

2.6.10 Other

The following approaches are not reflected in Table 2.3, because they do not propose definitions of but present theoretical perspectives on data center MRA fairness or show how to arrange fairness and efficiency requirements.

2.6.10.1 Fairness Quantification and Efficiency Tradeoffs

An area of fairness research in computer science is quantifying fairness of an existing allocation. Notably, this quantification is not even straightforward, when only a single resource has to be allocated. The question to be answered is, given an allocation vector $v \in \mathbb{R}^u_{\geq 0}$, where v_i is what user u_i receives, how fair is the allocation (quantified by a real number)? An early and prominent attempt to quantify fairness is Jain's Index [43] and a second approach is a-fairness [118]. The latter approach allows to specify a bounded fairness-efficiency compromise by parameter a. [51] determines a family of fairness measures by a function $f_\beta : \mathbb{R}^u_{\geq 0} \to \mathbb{R}$ parameterized by $\beta \in \mathbb{R}$. For certain values of β, f_β equates to Jain's index [43], a-fairness [118], or other prominent fairness measures, such as entropy [90]. Thus, [51] shows that these fairness measures lie on a continuum of functions, although they are conceptually different.

[44] develops a framework to address MRA fairness-efficiency tradeoffs under the assumption of Leontief utility functions. To this end, two families of fairness functions are developed, based on f_β. However, the two families are equal except that one familiy measures the bundle of a user u_i by u_i's dominant share and the other familiy measures u_i's bundle by the ratio of what u_i receives to u_i's resource requirement vector. Therefore, these families effectively operate on a scalar per user and not a vector, which constrains their applicability to MRA problems with Leontief utility functions. [107] extends the framework of [44] to multi-resource sharing in time to investi-

gate fairness-efficiency tradeoffs, when flows are scheduled based on multiple resources. [101] also addresses the fairness efficiency trade-offs for Leontief utility functions and presents an AP that achieves high utilization, while also ensuring SI, EF, and PE. As the allocation of multi-tiered storage is investigated, only two resources (hard disk and solid state drive) are allocated and these resources are directly comparable by their input/output operations per second.

2.6.10.2 General Framework

[41] theorizes a job scheduling framework, where the scheduler does not know the size of jobs until their completion. Even when all jobs are known in advance, instantiated problems are NP-hard. The formulation of the framework is generic, wherefore it can be applied to queuing and scheduling problems. Three classes of problems are discussed and it is shown that proportional fairness performs well for all of them. Due to the generality of the problem, the results are weak, according to the authors.

2.6.10.3 Fair Slowdown or Stretch

MRA fairness in data centers can also be defined as equalizing the slowdown (also known as stretch) that users experience, when the data center's load increases. This approach is, for example, taken by [18] to define fairness in desktop grids. Also [52] pursues this idea by considering the fairness of "progress shares" that "capture the contribution of each resource to the progress of a job". [98] adopts this fairness measure to determine a fair and efficient mixture for multiclass workflows.

While the slowdown can be considered a substitute for utility functions, which are not available in data centers, computing the slowdown requires to run workloads on an unloaded data center to determine a performance baseline to calculate the slowdown in a loaded data center [52]. Furthermore, the equal slowdown approach is not SI or EF [114]. Thus, defining multi-resource fairness as fair slowdown not only bears theoretical shortcomings but also practical overhead.

2.6.10.4 Tetris

[33, 34] present the multi-resource job scheduler Tetris, which shows how efficiency (in terms of makespan minimization) and fairness (according to different definitions) can be integrated in practice. The Tetris AP is implemented for Hadoop YARN and its primary goal is the minimization of makespan. Fairness is integrated by a knob mechanism: when resources become available, all pending tasks are sorted by a customizable fairness metric. The knob $k \in (0, 1]$ determines the fraction of the queue, from which the best fitting task in terms of resource utilization is selected. Thus, for minimal k efficiency is ignored to achieve the highest possible fairness. To determine how well a task fits, CPU, RAM, disk I/O, and network access of pending tasks are estimated by monitoring previous tasks of the same job. Tetris also addresses VE/task placement. The allocation of disk I/O and network access is controlled by token buckets, which is not detailed by any means.

2.7 Discussion

After formally defining multi-resource allocation problems, this chapter showed that multi-resource allocation problems arise in data centers (a) during scheduling, (b) on middle boxes, and when allocating node (c) PRs and (d) MRs. Utility functions suggested in literature to model these different allocation processes where discussed and single-resource allocation paradigms and their extension to MRA investigated. After discussing desirable characteristics for data center MRA, different approaches to data center MRA where discussed and compared in terms of dimensions discussed before hand. The discussion showed that establishing fairness in data centers by MRA is a complex task for the following reasons.

1. Fairness is to be established among users by allocating NRs to their VEs, whereat users operate different numbers of VEs.

2. Different heterogenous NRs, such as CPU, RAM, Disk I/O, network access, cache capacity, and memory bandwidth, are to be allocated, while dependencies among NRs differ from VE to VE and are fluctu-

ant over time and often unavailable due to technical or privacy constraints.

3. Heterogenous resources prohibit an objective comparison of resource bundles (this comparison is trivial, if there is only one resource), which complicates the definition of fairness.

4. VEs as well as NRs are partitioned over different nodes.

5. NRs are allocated in two steps: VE scheduling (which VE to start next) and NR allocation (how to allocate NRs to running VEs).

Many approaches achieve desirable characteristics (strategyproofness, sharing incentives, Pareto efficiency, or envy freeness). It was found that some of these characteristics are trivially achieved under those assumptions made by these approaches or that they are not relevant in data centers. Two important single-resource allocation paradigms are max-min fairness and proportional fairness. Both have been extended to multiple resources by different approaches and proportional fairness was proven to outperform max-min fairness in terms of efficiency under realistic assumptions.

Several approaches (DRF, PDF, PBS, and TSF) leverage exclusively VE scheduling to establish fairness in data centers. However, VE scheduling cannot control NR allocations directly. Particularly, when (a) the NR utilization of VEs is hard to predict, (b) highly fluctuant, and (c) VEs potentially run over very long periods, VE scheduling does not ensure that all users receive a fair amount of NRs. Other approaches focus directly on NR allocations (BBF and REF). However, these approaches are applied per node and the NR allocation among nodes is not coordinated. Furthermore, some approaches are unspecific in their fairness definition (RRF) or solely of theoretic interest (GRF). A further shortcoming of several approaches is that they prove their desirable characteristics on the assumption of Leontief utility functions (DRF and PDF), while Leontief utility functions are too simplistic in general. Also, it was found that some of these characteristics are trivially achieved under the assumptions made by these approaches or that they are not relevant in data centers.

Therefore, an approach is needed, that overcomes these shortcomings by adapting priorities of VEs to access PRs in a manner that is coordinated among nodes and intuitively fair among users. Particularly DRF, which has been extended in many directions, has been shown to not comply with an intuitive understanding of fairness. In addition, the approach to be developed must not make any assumptions on utility functions, because dependencies of PRs are unavailable due to technical or privacy constraints. It is also desirable, that this approach provides incentives to users to precisely specify the PRs their VEs will utilize, which will allow the data center operator to schedule VEs efficiently.

3

A Novel Fairness Definition for Clouds

Physical Resource (PR) allocation is the most potent step to control cloud resources and utility functions are the most important aspect, when theoretically modeling allocation problems. Because controlling cloud resources through PR allocation was a relatively unexplored research field before this thesis, this chapter begins by investigating which utility functions are applicable to model PR allocation in Section 3.1. The investigation shows that the dependency of PRs and their effects on VM performance are highly complex and virtually unpredictable. Unfortunately, the lack of a well-defined class of utility functions aggravates defining fairness. Thus, in order to find a fairness definition to be enforced among users of a private cloud, this lack has to be circumvented. This thesis does so by defining fairness as prioritizing consumers inversely to their greediness, which accounts for these technical constraints. Therefore, Section 3.2 investigates an intuitive understanding of greediness by a questionnaire among more than 600 individuals. Based on the questionnaire results, Section 3.3 develops a novel bundle measure termed the Greediness Metric (GM) that measures the greediness

of consumers, when they serve themselves from free-for-all resource pool. Cloud fairness is defined as prioritizing VMs inversely to the greediness of their owners, whereat owner greediness is determined by a refined GM. It is proven that this definition provides several desirable incentives to users.

3.1 Utility Functions of VMs

Utility functions (cf. Section 2.3) describe how combinations of resources influence the performance perceived by users. Therefore, utility functions (a) indicate in which ratios resources have to be allocated, in order to maximize user satisfaction and efficiency, and (b) are determined by technical factors. Thus, resource/performance dependencies and, accordingly, utility functions change depending on which resources are allocated. As PR allocation has so far rarely been investigated but is designated to be leveraged in this thesis, this section investigates PR/performance dependencies that exist during PR allocation in an attempt to conclude on a class of utility functions that is suited to model PR allocation. In particular, PR/performance dependencies are investigated by measuring the scores a VM achieves on different benchmarks and the PRs that the VM utilizes, while varying the VM's VRs and stress on the host system.

3.1.1 Methodology

This thesis investigates the nature of utility functions that VMs have during runtime, by examining how combinations of PRs, which are (a) made available to a VM and (b) utilized by the VM, affect this VM's performance [17, 111]. Therefore, different workloads are executed on a VM with a changing number of VCPUs and VRAM (this influences how many PRs the VM can access) and varying load levels of the host system (this simulates contention among VMs and also influences how many PRs the VM can access).

A machine with a 2.5 GHz AMD Opteron 6180 SE processor with 24 cores and 6 and 10 MB of level 2 and 3 cache, respectively, and 64 GB of ECC DDR3 RAM with 1333 Mhz is used as host system. This configuration was regarded a high-end data center node in 2011 and, thus, was a good representative of an average data center node, when the measurements were

conducted in 2014. VM and host have a x86-64 architecture and run Ubuntu 14.04.2 LTS, Trusty Tahr, which was the latest Ubuntu release, when the experiments were conducted. All reconfigurations of the VM as well as measurements were conducted by a python script.

3.1.1.1 Measurement Method

Every program is a process that consists of one or more threads, *i.e.*, a process is an accumulation of threads. Threads are the atomic resource consumers in computing environments. Therefore, if all possible resource consumption patterns that threads can exhibit are known, consumption patterns of processes (this includes VMs) can be deduced. However, measuring at the thread level is technically challenging. Furthermore, the resource utilization of a process is not simply the sum of resource utilization of its threads, because threads often share their process' memory or other resources. Therefore, measuring threads to conclude on the resource consumption of a process is unnecessarily complicated. Accordingly, resource consumption is measured directly at the VM level, as follows.

3.1.1.1.1 CPU time

The CPU time consumed by a VM is measured via its `qemu` [9] process. In particular, KVM in conjunction with qemu was used for virtualization, wherefore, a VM is a qemu process on the host system. The CPU utilization of this process is measured via `psutil`, which is a python module.

The VM's access to CPU time was altered by the VM's number of VCPUs (this requires restarting the VM) and by pinning the VM's VCPUs to physical CPU cores and also pinning stress tests [4] to these cores.

3.1.1.1.2 RAM

A significant amount of RAM is typically shared among processes. Most tools that measure RAM utilization of individual processes count such shared memory pages as if they were exclusively utilized by a process and, therefore, overestimate memory usage. To avoid such overestimation, the VM's RAM utilization is determined via `smem`'s Proportional Set Size (PSS) metric [92]. The PSS of a process p is defined by p's private memory plus the proportion of

shared memory pages, *i.e.*, the more processes share a memory page, the less this page increases *p*'s PSS. However, it is possible that the RAM utilization measured for a VM slightly exceeds the VM's VRAM (cf. Figures 3.1, 3.2, and 3.3), because the VM process also owns RAM pages that are (a) necessary for virtualization and emulation and (b) independent of the VM's VRAM (those pages cannot be used by the guest OS as RAM).

The VM's RAM access is altered by changing the VM's VRAM. Just as changing the VM's VCPU this requires restarting the VM.

3.1.1.1.3 Permanent Storage I/O

The VM's permanent storage I/O is measured via `psutil` in read/write operations and bytes read/written. However, as most of the VM's permanent storage content is cached in the host's RAM, the VM's permanent storage I/O is insignificant and, therefore, not discussed subsequently. For the same reason, the VM's permanent storage I/O is not altered. [97] studies the role permanent storage I/O plays for a VM's utility function in more detail.

3.1.1.1.4 Network Access

VMs are connected to the physical network via a virtual network interface. Thus, the VM's network traffic is measured via the traffic on this virtual interface. However, all tested workloads are local, where they do not produce traffic on this interface. For the same reason, network access is not altered in the experiments. [97] studies the role network access plays for a VM's utility function in more detail.

3.1.1.2 Workloads

Workloads are simulated by benchmarks of the Phoronix test suite [76]. Using benchmarks to simulate workloads has the advantage, that VM performance, *i.e.*, utility, is clearly captured by the benchmark score. A program/benchmark is called a "single-core" program/benchmark, if it only has a single thread, *i.e.*, cannot utilize multiple CPUs in parallel. In contrast, a program/benchmark is called a "multi-core" program/benchmark, if it is multi-threaded, *i.e.*, can utilize multiple CPUs in parallel. The following benchmarks were selected, to represent workloads that are often executed in clouds.

3.1.1.2.1 Apache

Clouds frequently host web servers and Apache is the globally most popular web server software. Therefore, the Apache benchmark is selected as a workload. This workload measures how many requests the Apache server can sustain concurrently. Therefore, the higher the score, the better.

3.1.1.2.2 Nginx

The second most popular webserver is nginx. Just as the Apache benchmark, this benchmark measures how many requests the nginx server can sustain concurrently. Therefore, the higher the score, the better.

3.1.1.2.3 Aio-stress

The speed of permanent storage I/O (hard disk or solid state drive) is critical for the performance of a large variety of applications. Therefore, this speed is assessed by the aio-stress benchmark. The higher the score, the better.

Data a VM believes to reside in permanent storage may actually be cached in RAM by the hypervisor. Therefore, executing this benchmark or any permanent storage access in a VM, not necessarily results in accessing the physical permanent storage (cf. Section 3.1.1.1.3).

3.1.1.2.4 Phpbench

Older websites may be based on PHP. Therefore, `phpbench` is selected as benchmark as well. phpbench performs 10^6 (simple) tests in order to bench various aspects of the PHP interpreter. The higher the score, the better.

3.1.1.2.5 Pybench

Python is a popular scripting language. Therefore, the `pybench` benchmark was selected to represent a workload. Pybench measures the execution time of Python functions such as BuiltinFunctionCalls and NestedForLoops. Therefore, the lower the score, the better.

3.1.1.2.6 7zip

Data is compressed to reduce storage or traffic requirements. Therefore, compression is a common task in clouds and a wide range of compression

algorithms exists. The 7zip benchmark was chosen as the representative of compression tasks in these investigations. The benchmark uses 7zip's integrated benchmark feature and a higher score means better performance.

3.1.2 Results

This thesis achieved the results presented below by deploying the setup above to investigate how VM performance changes depending on the amount of RAM and CPU the VM has access to and how the shortage of one of those PRs influences the utilization of the other PR. Also, it was investigated how stress on the host system influences the VM's performance.

3.1.2.1 RAM

The VRAM of a VM is altered and the effects on the VM's performance observed [8]. The smallest amount of VRAM is 100 MB, because less VRAM results a kernel panic while booting the VM. Although more VRAM allows a VM to utilize more RAM, the VM not necessarily utilizes all RAM available. Therefore, the performance is not only contrasted with the VM's VRAM but also with the RAM the VM maximally utilizes over its runtime. For each VRAM configuration 10 measurements are conducted. The subsequent subsections measure performance by selected benchmarks. Before the benchmark is started, the VM utilizes 230 MB of RAM (if its VRAM allows to utilizes this amount).

3.1.2.1.1 Apache

Figure 3.1 shows the Apache scores achieved and the amount of RAM utilized by a VM depending on different amounts of VRAM. When the VM executes Apache, the VM process never utilizes more than 390 MB of RAM. In particular, for a VM with 100 to 350 MB of VRAM the amount of RAM that is maximally utilized continuously increases but does not further increase, when more than 350 MB of VRAM are added. Therefore, Figure 3.1 shows that a VM with less than 350 MB of VRAM utilizes all RAM that is available, which seems to imply, that this amount of RAM is critical for performance. However, Figure 3.1 also depicts that the Apache score only increases for up to 250 MB of VRAM and that this increase is marginal compared to the in-

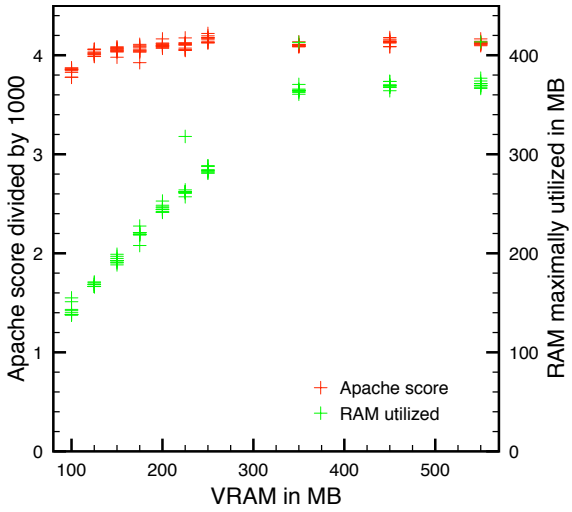


Figure 3.1: Apache scores achieved and amount of RAM utilized depending on the amount of VRAM

crease of RAM that is utilized. Therefore, the dependency between VRAM and utilized RAM is much stronger than the dependency between VRAM/utilized RAM and Apache score. In particular, while the RAM utilization more than doubles, the Apache scores vary by less than 10%. This is particularly interesting, because this configuration range includes 100 MB of VRAM which constrains the VM's RAM utilization to less than half of what the VM alone (without executing any workload) would utilize.

3.1.2.1.2 PHP

Figure 3.2 shows that when the VM executes phpbench, the VM process utilizes 270 MB of RAM at most. Although the VM is constraint in its RAM utilization, when it has less than 250 MB of VRAM, there is no correlation between the achieved phpbench score and the VM's VRAM, as the phpbench score does not increase.

3.1.2.1.3 Python

Figure 3.3 shows the same results as Figure 3.2: the VM process utilizes 270 MB of RAM at most and even when the VM is constraint in its RAM utiliza-

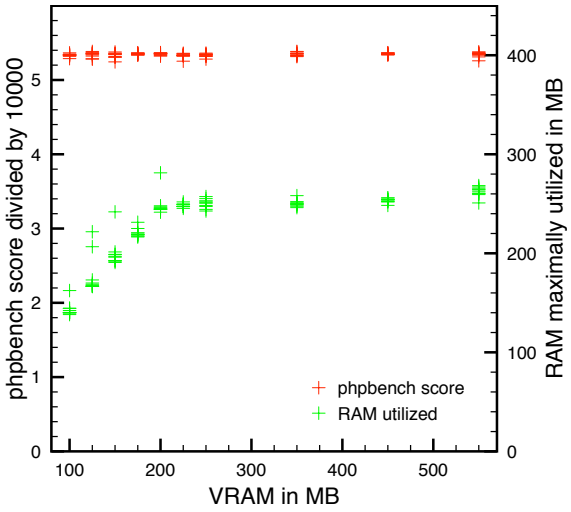


Figure 3.2: phpbench scores achieved and amount of RAM utilized depending on the amount of VRAM

tion, the pybench score does not suffer. Therefore, also the phpbench score is independent of the VM's VRAM.

3.1.2.1.4 Findings

RAM, which is actively utilized by a VM (be it on startup or when executing an application), not necessarily impacts the VM's performance. In particular, even if the RAM utilized by a VM varies from 100 MB to 350 MB, the VM's Apache score, *i.e.*, its ability to sustain concurrent server requests, only changed by 10%. For other benchmarks (phpbench and pybench) the score was entirely independent of the available RAM. This is particularly interesting, because not even a VM with 100 MB of VRAM showed decreased performance, while this is the minimum amount of RAM that avoids a kernel panic and even a VM that not executes any workload utilizes more, if possible.

3.1.2.2 CPU

Effects of increasing the number of VCPUs and stressing the host CPU are discussed next.

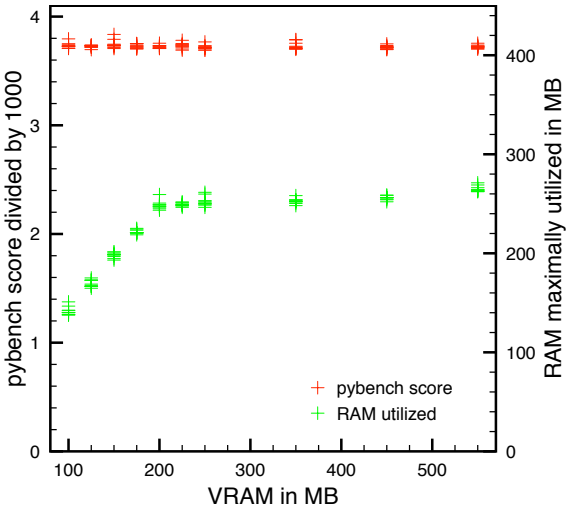


Figure 3.3: pybench scores achieved and amount of RAM utilized depending on the amount of VRAM

3.1.2.2.1 Multi Core Penalty

Increasing the number of VCPUs of a VM can decrease its performance. In particular, Figure 3.4 plots the Apache scores achieved by a VM with 1 to 9 VCPUs, whereat 16 measurements per configuration were conducted. The figure shows that the best performance is achieved, when the VM has three or four VCPUs, while additional VCPUs linearly decrease the Apache score. As the figure depicts, up to three VCPUs significantly increase performance and four VCPUs perform equally well. However, adding additional VCPUs continuously decreases performance. This effect, which is termed *multi-core-penalty*, occurred independent of whether VCPUs were pinned to physical CPUs. Figure 3.4 also demonstrates that, while three VCPUs perform best for an unstressed host, two VCPUs perform best, when the host is stressed.

Figure 3.4 also shows the Apache score, when the benchmark is executed natively, *i.e.*, directly on the host and not inside a VM. The multi-core-penalty does not occur for this native execution, which shows that it is caused by the virtualization layer. However, also when executed natively, the Apache score does not increase for more than seven cores but starts to scatter more.

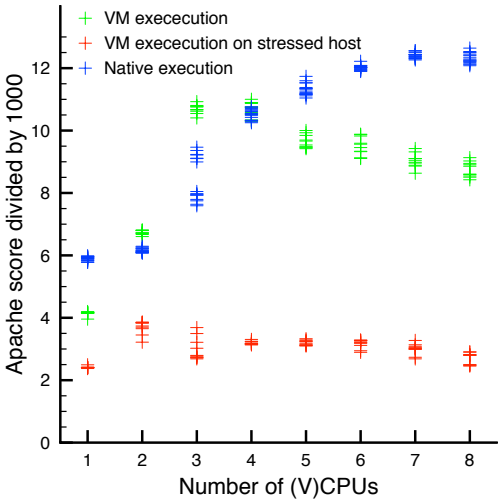


Figure 3.4: Apache scores achieved with different numbers of cores by native or virtual execution

What remains unclear is why a VM with three VCPUs clearly outperforms the native execution. Also it has to be investigated, if the multi-core-penalty only occurs for the KVM or also other hypervisors. This effect is not to be confused with the well known problem of virtualization overhead. See [25] for more details.

Figure 3.5 illustrates a similar effect for the aio-stress benchmark, where a VM with one VCPU constantly achieves a higher aio-stress score than any VM with more VCPUs. In particular, the aio-stress score of a VM with only one VCPU is on average a 30% higher than the aio-stress score of VMs with more VCPUs. However, unlike the Apache benchmark, the aio-stress score does not decrease further with the number of VCPUs.

As discussed above, more than four VCPUs result in lower Apache scores. Despite this decrease, the VM's utilization of CPU time increases with the number of VCPUs. In particular, Figure 3.6 shows that for 9 VCPUs the utilized CPU time is roughly twice as high as the CPU time utilized by one to three VCPUs. This figure also indicates that three VCPUs actually need less CPU time than two VCPUs (while achieving a significantly better Apache score than two VCPUs), when the host system is not stressed. More impor-

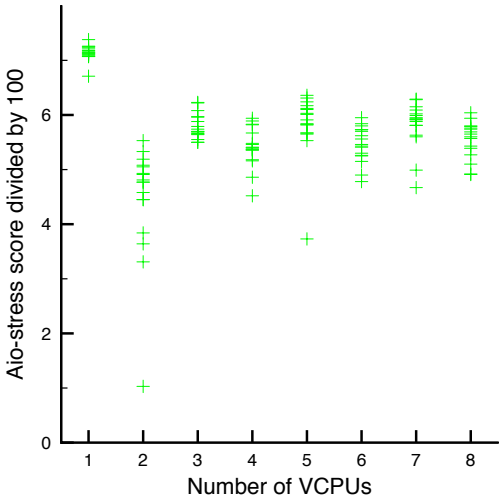


Figure 3.5: aio-stress scores achieved by VMs with varying numbers of VCPUs

tantly, one and two VCPUs seem to be more moderat with respect to CPU time usage (9% less), when a stressed host is compared to a not stressed host. However, this trend inverses for four and more VCPUs, *i.e.*, stress on the cores the VCPUs are pinned to also increases the CPU time requirements by roughly 8%.

Therefore, the utilized CPU time does not correlates with the VM performance in a distinct way.

3.1.2.2.2 Multi Core Advantage

The multi-core-penalty shows that multi-core workloads may perform worse, when executed in a VM with multiple VCPUs. Similarly, the performance of single-core workloads can benefit, when executed in a VM with multiple VCPUs. In particular, this occurs, when the host system is under stress.

As an example, Figure 3.7 illustrates the nginx scores achieved with different numbers of VCPUs. The figure shows that two VCPUs perform significantly better than one VCPU but adding more than three VCPUs does not increase performance. However, every additional VCPU increases the

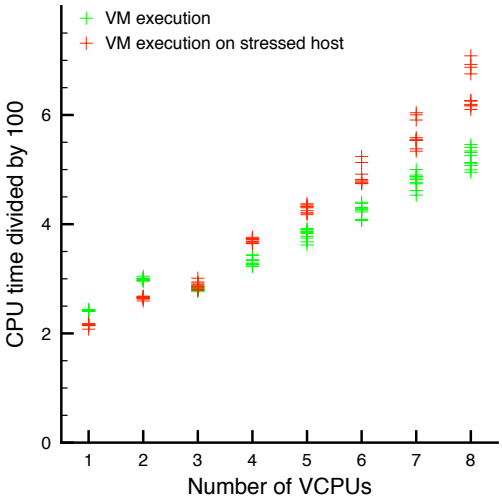


Figure 3.6: CPU time utilized by VMs with varying number of VCPUs executing the Apache benchmark

nginx score, when the host is stressed. Nonetheless, the scores achieved on a stressed host are significantly lower in general.

The reason for this increase is that the VM runs as a (qemu) process and the entitlement to CPU time of this process increases with the number of VCPUs. When CPU time is under contention, this entitlement is taken into account to allocate CPU time to contending processes. This mechanism is termed *multi core advantage*.

Figure 3.8 shows the multi core advantage for phpbench. In particular, the phpbench score is the same for all numbers of VCPUs, when the host is not stressed. When the host is stressed, the VM achieves this "unstressed performance" for four or more VCPUs. For pybench the multi core advantage also occurred. However, for pybench already two VCPUs achieve the unstressed performance.

3.1.2.3 Dependency of CPU and RAM

After investigating the effect of CPUs and RAM on performance, the dependency of these two PRs is investigated.

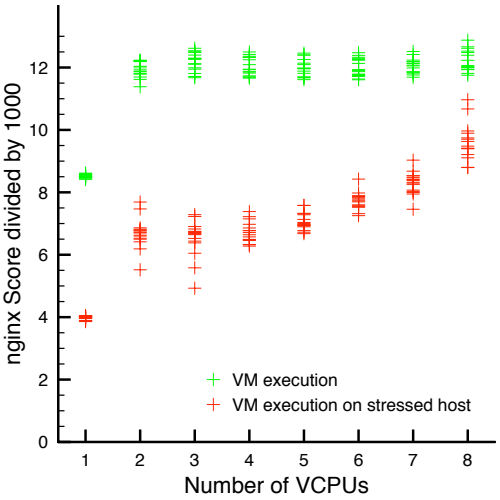


Figure 3.7: nginx scores achieved by VMs with varying numbers of VCPUs

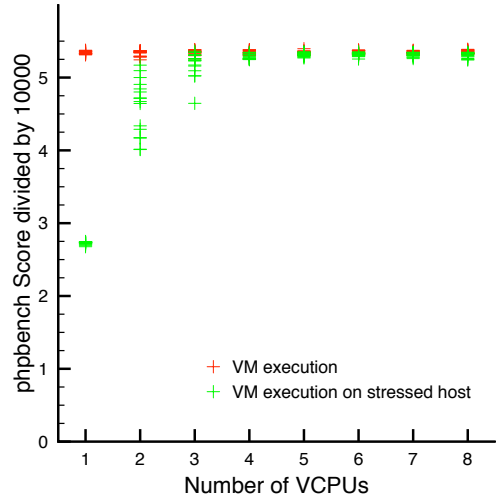


Figure 3.8: phpbench scores achieved by VMs with varying numbers of VCPUs

3.1.2.3.1 VCPUs and Maximal RAM Utilization

No dependency between CPU and RAM usage was found in general. However, the 7zip benchmark revealed an interesting dependency of VCPUs and RAM utilization (cf. Figure 3.9). As Figure 3.9a shows, for one to three VCPUs a VM executing the 7zip benchmark utilizes 1 GB of RAM and for every two additional cores the RAM utilization increases by 400 MB (the VM had 9 GB of VRAM).

The distinct pattern in which RAM is utilized gives reason to believe, that it is essential for performance. Therefore, Figure 3.9b compares the 7zip scores achieved by VMs with 1 and 9 GB of VRAM. As Figure 3.9a shows, the more VCPUs a VM has, the more it will be constrained by only having 1 GB of VRAM, while 9 GB of VRAM not even constrain a VM with 24 VCPUs. In line with this observation, Figure 3.9b shows that the difference between the 7zip scores achieved by VMs with 1 and 9 GB of VRAM grows with the number of VCPUs. However, the score difference is rather moderate compared to the large difference in terms of RAM utilization. In particular, a VM with 24 VCPUs utilizes more than 5 GB of RAM, if available. This is five times as much, as a VM with 1 GB of VRAM utilizes. However, the 7zip scores achieved by these VMs only differ by 15%.

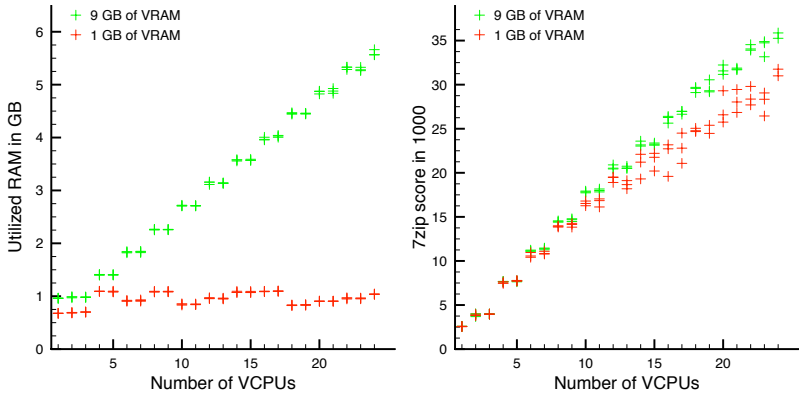
3.1.2.3.2 CPU time and RAM Utilization Pace

Several benchmarks increase RAM utilization steadily over time to a certain point. The pace with which this point is reached, often depends on the CPU time that the VM receives. For example, when a VM received 50% of the CPU time that it would receive from an unstressed host, the pace with which RAM was utilized also decreased by 50%.

Therefore, while the number VCPUs may influence the maximal RAM usage (cf. Section 3.1.2.3.1), the speed with which RAM is utilized depends on the CPU time received from these VCPUs.

3.1.2.4 New Findings

Most work on data center resource allocation assumes that resources such as CPU and RAM are required in static or at least well defined ratios and that



(a) RAM utilization

(b) VM performance measured by the 7zip score

Figure 3.9: RAM utilization and performance, depending on the number of VCPUs and amount of VRAM, of a VM executing the 7zip benchmark

the resulting performance is clearly defined (cf. Chapter 2). The results of this section do not confirm these idealistic assumptions.

Section 3.1.2.1 did not find any significant effect of VRAM on VM performance. Notably, even for workloads that seem to be RAM critical, as they utilize RAM in distinct patterns, or workloads running on VMs with just enough VRAM to avoid a kernel panic during boot, no significant effect was found. Section 3.1.2.3.1 showed that even if a lack of RAM impedes performance, the impediment is minor compared to the amount of RAM that is missing. In contrast, a lack of RAM bandwidth significantly effects performance [35] but is rarely considered, when investigating data center fairness. Another counter-intuitive finding is that when multi-core benchmarks are executed inside a VM, the performance often decreases, when more VCPUs are added to the VM (cf. Section 3.1.2.2.1). In contrast, single-core benchmarks profit from additional VCPUs, when the host is under stress (cf. Section 3.1.2.2.2).

Section 3.1.2.3 showed that the amount of RAM that is utilized by a VM may depend on the number of VCPUs and that the speed with which it is utilized depends on the CPU time allocated to the VM's VCPUs.

Therefore, it is concluded that no assumption on utility functions can be made, when defining cloud fairness.

3.2 Fairness and Greediness Questionnaire

The dependency of different resources, when utilized by VMs, and their effects on VM performance are (a) highly complex, (b) fluctuating, and (c) also depend on the workload that is executed by the VM. Therefore, assumptions on utility functions in clouds cannot be made. This makes it impossible to apply traditional fairness definitions that rely on assumptions on utility functions. Also assumptions on a well structured negotiation process cannot be made, as VMs utilize PRs of their host as needed (within the hard limits defined by their VRs) in a highly fluctuating manner. In particular, a node can be thought of as "self-serving buffet" for the VMs that are hosted, as VMs request PRs and the node provides for these PRs, if available. Since the host has the possibility to deny or prioritize VM requests, fairness has to be introduced by rationing PR access, when overload occurs. Thus, cloud fairness has to be defined via a prioritization mechanism that constrains VM access to PRs, when PR overload occurs, such that all users are have access to a fair amount of PRs.

Therefore, when abstracted from the cloud context, fairness has to be defined by answering the following question: *when different consumers access a free-for-all resource pool and overload on some resources occurs, which consumers must be constrained in order to enforce fairness*? The premiss of this thesis is that it is fair to constrain consumers in proportion to their greediness, *i.e.*, the greedier a consumer serves herself from the resources pool, the more her access must be constrained in favor of other consumers. This, in turn, raises the question of *how the greediness of consumers can be defined and quantified based on their multi-resource self-servings*? When this question can be answered, an allocation is fair, when (a) this greediness quantification is aligned for all consumers and (b) greedy consumers are constrained in favor of less greedy consumers.

To thoroughly investigate this question, a questionnaire was developed to evaluate an intuitive understanding of fairness an greediness. DRF's confor-

mance with an intuitive understanding of fairness was also evaluated in this questionnaire, since (a) DRF is the most prominent approach for fairness in data centers (cf. Section 2.6.1) and (b) respective replies serve as checks for questionnaire reply qualities. Therefore, this questionnaire allows to derive a justified and intuitive understanding of fairness and greediness, for the case where little or no information about consumers' utility functions is available. Avoiding any explicit information on utility functions was decided on purpose, because in case of PR allocation, utility functions are not known (cf. Section 3.1). Appendix A.1 shows the original questionnaire in full and describes how participation was requested.

The questionnaire specified real-life scenarios in terms of three questions Q_1, Q_2, and Q_3 to not distract participants by technical terms and let them fully concentrate on the question of fairness. For example, Q_3 describes a scenario, where three bakers purchase together three ingredients for cakes. The bakers prepare different recipes and split the ingredients by putting them on a table from which every baker serves herself. This was done until at least one resource on the table was depleted, which prohibited the utilization of the other resources. While this is a specific scenario the underlying problem is generic: Common heterogenous resources are split among individuals with different demands by letting them serve themselves. This implies mutual trust and poses the question of how individuals, who could try to exploit the system, can be identified, that is, how disproportionate consumption can be defined. The transferability to clouds is evident: just as the bakers, VMs serve themselves from a pool of common resources (the PRs of their host).

All questions were designed in order avoid any selection bias. For example, in Q_3 participants were asked to establish a ranking over the three bakers and a bias was avoided by allowing participant to create this ranking via a drag and drop menu, as shown in Figure 3.10. In contrast to a list of all rankings, where some rankings appear first and are, therefore, more likely to be selected, this drag and drop menu made it equally easy to select any ranking in an intuitive manner.

Questionnaire participants had to chose between different options of allocations or define rankings of consumers. Additionally, participants were offered to explain their answers in text boxes. The questionnaire did not ad-

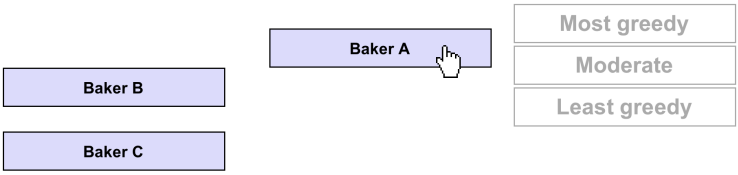


Figure 3.10: Drag and Drop menu with which participants had to specify their answer to Q3

dress any particular target group. Out of 721 participants, who started the questionnaire, 604 completed it.

3.2.1 Choosing the Most Fair Allocation (Q1)

Q1 was designed to evaluate a key building block of DRF: the use of the L_∞ norm to measure the value of a bundle, *i.e.*, that the value of a bundle is defined solely by the resource that it contains in the largest proportion (cf. Section 2.6.1). The scenario described covers two resources r_1 and r_2 of which six and twelve units where available, respectively. These resources have to be allocated to three consumers c_1, c_2, and c_3. c_1 only requires r_1, c_2 only r_2, and c_3 requires for each unit of r_1 two units of r_2. This results in seven possible allocations to allocate all resources and do not give consumers resources they have no use for. However, most of these allocations are intuitively unfair, *e.g.*, in two of these allocations at least one consumer receives no resources at all. Because the scenario describes that resources are requested in static ratios, it is transferable to allocation problems in data centers, where these Leontief utility functions (cf. Section 2.3.3) are the standard assumption [23, 30].

Participants were presented with four of the seven possible allocations and asked to choose *the* allocation that seemed most fair to them. The allocations to choose from were presented numerically and graphically to participants (cf. Appendix A.1.1). Table 3.1 shows the allocations participants had to choose from and the respective labels (allocation A11, A12, A13, and A14). Figure 3.11 shows that A11 and A14 were only chosen by a minority of the 721 participants (0.4% and 1.1%, respectively) and most participants deterred between A12 and A13 (30.0% and 68.5%, respectively). The three allocations that were not displayed were even more extrem in terms of re-

Table 3.1: The four options A11, A12, A13, and A14 in Q1 of the questionnaire to allocate the two resources r_1 and r_2 to the three consumers c_1, c_2, and c_3

	A11		A12		A13		A14	
Consumer	r_1	r_2	r_1	r_2	r_1	r_2	r_1	r_2
c_1	2	0	3	0	4	0	5	0
c_2	0	4	0	6	0	8	0	10
c_3	4	8	3	6	2	4	1	2

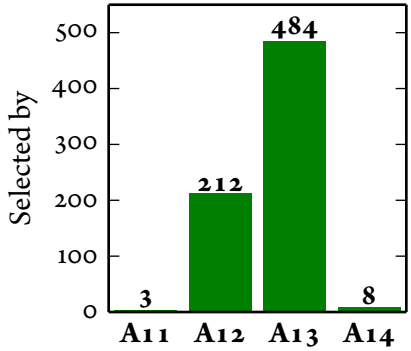


Figure 3.11: Number of participants who selected the different answers to Q1

source distribution than A11 and A14. As the latter two allocations received very low acceptance, those three allocations left out would have been selected by even less participants. Therefore, leaving out those three allocations did not create a selection bias but instead allowed participants to focus on the important choice between A12 and A13.

The arguments below support A12 and were frequently given by participants in the text boxes that allowed to explain their answer.

- c_1 and c_2 only compete with c_3 for resources, but not with one another. A fair allocation splits resources equally between those who contend for them.

- All receive an equal amount of what they want.
- This is the only allocation where nobody can complain that someone has more of the same resource.
- All consumers have equal utilities.
- When prices are introduced based on available units, this option gives the same value to all consumers.

The arguments below support A13 and were frequently given by participants in the text boxes that allowed to explain their answer.

- When prices are introduced based on available units, this option gives the same value to all consumers.
- c_1 and c_2 receive $2/3$ of *one* resource and c_3 $1/3$ of *two* resources, which makes $2/3$ for everybody. On a similar note, some participants rejected A12, because c_3 gets as much as c_1 and c_2 combined.
- Because c_1 and c_2 only want one resource, they should get more of it than c_3, as c_3 wants both resources.
- This option is the result of a simple auction or when all consumers get an equal share of both resources and then trade.
- The range of numbers of units given to consumers is the smallest. In particular, for A13 the range is 4-6-8, while it is 3-6-9 for A12.

3.2.2 Allocating Based on Resource Requests (Q2)

Q2 was designed to validate how participants allocate scarce resources based on requests. This problem is relevant in clouds, because, when a host experiences congestion, it has to decide which requests to accept and which to delay or reject. In Q2 three resources r_1, r_2, and r_3 (for each eight units were available) had to be split between the two consumers c_1 and c_2 (cf. Appendix A.1.2). All three resources were initially contended, but the contention for r_1 can be resolved in bilateral negotiations, with c_1 receiving two units and c_2

Table 3.2: The three options A21, A22, and A23 in Q2 of the questionnaire to allocate the three resources r_1, r_2 and r_3 to the two consumers c_1 and c_2

	A21			A22			A23		
Consumer	r_1	r_2	r_3	r_1	r_2	r_3	r_1	r_2	r_3
c_1	2	4	4	2	5	5	2	4	5
c_2	6	4	4	6	3	3	6	4	3

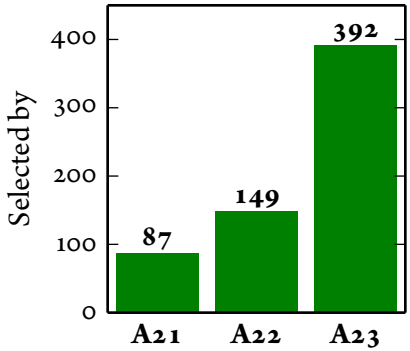


Figure 3.12: Number of participants who selected the different answers to Q2

six. However, these negotiations left open how r_2 and r_3 had to be allocated. c_1 demanded five units of both resources and c_2 demanded four units of both. Participants had to decide, whether one consumer has to cease both missing units or each consumer should cease one. Accordingly, the most fair option had to be chosen from allocations A21, A22, and A23 as shown in Table 3.2.

628 participants answered this question, as illustrated in Figure 3.12. From the information given by participants in the text boxes, the following frequent arguments were compiled, indicating the subjective perception of fairness.

A21 was supported by 13.9% of the participants. The arguments below were most frequently given by those participants to justify their choice.

- Since r_1 is not under contention anymore, it does not need to be considered for the allocation of r_2 and r_3. Consequently, c_1 and c_2 should get a fair, that is, equal, amount of r_2 and r_3.

- Because c_2 receives more of the first resource, c_2 will probably also require more of the other resources, and thus should not be constrained on these.

- The consumer, who demands the greater amount of a contented resource, should cease the deficit, because relative to his demand this deficit is smaller than for someone who demands less.

A22 was supported by 23.7% of the participants. The argument below was most frequently given by those participants to justify their choice.

- Because c_1 gets 4 units less of the first resource compared to c_2, c_1 should get four units more of the other resources.

A23 was supported by 62.4% of the participants. The argument below was most frequently given by those participants to justify their choice.

- Since r_1 is no more under contention, it does not need to be considered for the allocation of r_2. Assuming that the demands reflect actual needs (rather than negotiating positions), the most fair thing to do is splitting the scarcity equally, *i.e.*, both get one resource less than requested.

3.2.3 Estimating Greediness (Q3)

Q3 was designed to collect information on how the greediness of consumers, who served themselves from a pool of common resources, is perceived. In addition, insights are collected on how proportionality and value of resource bundles is perceived, when no information about consumers' utility functions is available. Thus, Q3 is most important for the investigation of an intuitive fairness understanding, as it provides insights on how resources that different VMs on the same host utilize can be compared.

Q3—as defined in Appendix A.1.3—is based on three scenarios S31, S32, and S33, were three consumers c_1, c_2, and c_3 had served themselves from a

Table 3.3: The three scenarios S31, S32, and S33 in Q3 of the questionnaire, where three consumers c_1, c_2, and c_3 served themselves from a pool of three common resources r_1, r_2 and r_3

	S31			S32			S33		
Consumer	r_1	r_2	r_3	r_1	r_2	r_3	r_1	r_2	r_3
c_1	4	3	3	4	2	4	4	1	4
c_2	2	1	5	1	4	3	1	4	3
c_3	4	2	1	1	6	2	1	6	2
Remainder	2	3	0	6	0	0	6	1	0

pool of three common resources r_1, r_2, and r_3 (like VMs on the same host). To split these resources, each consumer had allocated himself a certain bundle as shown in Table 3.3. The three consumers had to be ranked according to how their greediness was perceived, all being based on the amounts the consumers had allocated themselves.

3.2.3.1 Metrics

Many participants tackled Q3 by proposing a metric to assess the value of bundles. These metrics encompasses the following four: cost, cost$\times$scarcity, cost$\cap$scarcity, and DoRe. All of those metrics are intuitive, as they have a straight-forward informal and formal definition and were suggested by participants with no expert background. While this intuitiveness is a merit of these metrics, all of them have drawbacks, as was identified by the questionnaire and is discussed in Sections 3.2.4.3 and 3.2.4.4.

3.2.3.1.1 Cost

The cost metric is the simplest metric and was often suggested by participants. The value of one unit of resource r_i is defined as $p/\overleftrightarrow{r_i}$, where p is a constant and $\overleftrightarrow{r_i}$ is the number of units available of r_i. The value of a bundle is the sum of values of its resources. For example, for $p = 1$ the value of c_2's bundle in S31 is

$$\frac{2}{12} + \frac{1}{9} + \frac{5}{9} = \frac{5}{6}.$$

This metric is equivalent to the the L_1 norm and the sum-based-penalty function presented in Section 2.6.7 and [48]. The term *cost*-metric chosen here reflects the wording frequently chosen by questionnaire participants, who suggested this metric.

3.2.3.1.2 Cost×scarcity (C×S)

The cost×scarcity metric is a natural extension of the cost metric. The value of one unit of resource r_i is defined as $a(r_i) \cdot p / \overleftrightarrow{r_i}^2$, where $a(r_i)$ is the amount that is allocated in total of r_i. The value of a bundle is defined as the sum of values of those resources contained. For example, for $p = 1$ the value of c_2's bundle in S31 is

$$\frac{2 \cdot 10}{12^2} + \frac{1 \cdot 6}{9^2} + \frac{5 \cdot 9}{9^2} = \frac{29}{54}.$$

3.2.3.1.3 Cost∩scarcity (C∩S)

The cost∩scarcity metric is another natural extension of the cost metric and defines the value of a resource just as the cost metric. However, the value of a bundle is defined only over resources that are depleted, *i.e.*, resources where $a(r_i) = \overleftrightarrow{r_i}$. For example, for $p = 1$ the value of c_2's bundle in S31 is

$$\left\lfloor \frac{10}{12} \right\rfloor \cdot \frac{2}{12} + \left\lfloor \frac{6}{9} \right\rfloor \cdot \frac{1}{9} + \left\lfloor \frac{9}{9} \right\rfloor \cdot \frac{5}{9} = \frac{5}{9}.$$

3.2.3.1.4 DoRe

DRF defines the value of a bundle by the L_∞ norm, *i.e.*, by the biggest share of any resource relative to the overall amount of the resource, which is captured here by the DoRe (DOminant REsource) metric. For example, the value of c_2's bundle in S31 is:

$$\max\left(\frac{2}{12}, \frac{1}{9}, \frac{5}{9}\right) = \frac{5}{9}.$$

According to the L_∞ norm, the bundles of c_1 and c_2 in S33 are equally valuable. This tie is broken by considering the second biggest share in the bundles, wherefore c_1's is more valuable.

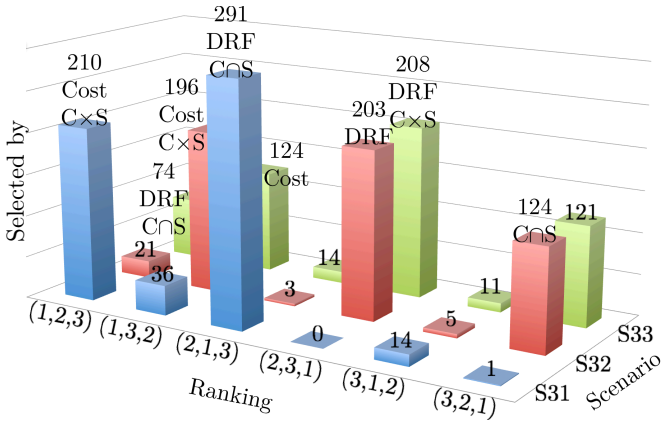


Figure 3.13: Number of participants who selected the different rankings (represented by triplets) in the three scenarios of Q3

3.2.3.2 Frequency Investigations

In this section numerical results of Q3 are presented. The implications of these results are discussed in Sections 3.2.4.3 and 3.2.4.4. For some rankings the free text indicated that the participant had assumed real life values of the resources and ranked the consumers accordingly. These rankings, as well as, incomplete rankings were removed, wherefore the presented results are based on 552 answers.

Subsequently, consumer rankings are denoted by triplets. For example, when the first consumer is ranked moderate, the second consumer is ranked most greedy, and the third consumer is ranked least greedy, this is denoted by the triplet (2,1,3).

Figure 3.13 illustrates for each scenario how many participants selected each ranking and highlights the rankings that correspond to the metrics discussed in Section 3.2.3.1. Table 3.4 shows which ranking corresponds to which metric for the three scenarios. Furthermore, Table 3.4 provides the percentage of participants, that answered with the respective ranking (deduced directly from numbers of Figure 3.13). Because these rankings given by most participants do not match the same metric over the three scenarios, Table 3.5 shows the quantity of the sixteen most prominent combinations of

Table 3.4: Percentages of most frequent rankings in Q3

Metric	S31	S32	S33
DoRe	(2,1,3): 52.7%	(2,3,1): 36.8%	(2,3,1): 37.7%
Cost	(1,2,3): 38.0%	(1,3,2): 35.5%	(1,3,2): 22.5%
C×S	(1,2,3): 38.0%	(1,3,2): 35.5%	(2,3,1): 37.7%
C∩S	(2,1,3): 52.7%	(3,2,1): 22.5%	(1,2,3): 13.4%

Table 3.5: Quantity of participants who selected the most frequently selected combinations of rankings in Q3

Quantity	S31	S32	S33	Conforming with
79	(2,1,3)	(2,3,1)	(2,3,1)	DoRe
55	(1,2,3)	(1,3,2)	(2,3,1)	C×S
43	(1,2,3)	(1,3,2)	(1,3,2)	Cost
32	(2,1,3)	(3,2,1)	(1,2,3)	C∩S
30	(2,1,3)	(1,3,2)	(1,3,2)	
28	(2,1,3)	(3,2,1)	(3,2,1)	
27	(1,2,3)	(3,2,1)	(3,2,1)	
24	(2,1,3)	(2,3,1)	(3,2,1)	
22	(1,2,3)	(2,3,1)	(2,3,1)	
20	(1,2,3)	(2,3,1)	(3,2,1)	
18	(2,1,3)	(2,3,1)	(1,3,2)	
17	(2,1,3)	(1,3,2)	(2,3,1)	
16	(2,1,3)	(2,3,1)	(1,2,3)	
12	(1,3,2)	(1,3,2)	(1,3,2)	
9	(2,1,3)	(1,3,2)	(1,2,3)	
8	(2,1,3)	(3,2,1)	(2,3,1)	

rankings (which cover 79.6% of all participants). In addition to these combinations, three combinations were selected five and seven times each, two combinations were selected four and six times each, 4 combinations were selected three times, and 15 combinations were selected one and two times each.

3.2.4 Discussion

This section draws conclusions from the questionnaire's results. Section 3.2.4.1 concludes that A13 is the intuitively fair allocation, as it found significantly higher acceptance among participants than A12 and some arguments given in support of A12 were false. Section 3.2.4.2 proves that most participants consider it fair to allocate the deficits and not the actual resources fair, even when the deficit is calculated from requirement statements that cannot be verified. Unfortunately, such mechanism gives incentives to consumers to exaggerate demands and is, therefore, not applicable in many real world scenarios, such as cloud resource allocation. Section 3.2.4.3 points out that the rankings given by participants often are not consistent with the same metric over all three scenarios and analyses participants' arguments to conclude on how they made a decision. Section 3.2.4.4 argues that all metrics have disadvantages. This includes the DoRe metric, which achieves the highest possible compliance with participants' rankings. Its insufficiency stems from the facts that it (a) achieves this high compliance for the wrong reasons, *i.e.*, it works differently than participants argued, wherefore examples can be constructed where it results in rankings that contradict the arguments of participants, and (b) fails to identify A13 of Q1 as most fair allocation.

3.2.4.1 Choosing the Most Fair Allocation (Q1)

Only the first two arguments in favor of A12 are correct. The third argument (nobody can complain that someone has more of the same resource) is incorrect, because c_1 receives least of r_2 and c_2 least r_1. The fourth argument (all consumers have equal utilities) lacks foundation, because nothing was said about utilities of consumers, only about ratios in which they request resources. The fifth argument states that all receive the same value, which is actually the case for A13. Therefore, the arguments given in text boxes in

favor of A13 are more versatile and credible than those in favor of A12. Because A12 and A13 were chosen by 30.0% and 68.5% of the participants, respectively, that is, A13 was chosen more than 2.25 times as often as A12, it is concluded that A13 determines the intuitively fair allocation.

The DRF allocation is A12. Therefore, this result shows that DRF, which is the de facto standard in data center multi-resource fairness (cf. Section 2.6.1), does not comply with an intuitive understanding of fairness. This discussion is deepened in Section 3.2.4.4.

3.2.4.2 Allocating Based on Requests (Q2)

While supporters of A21 and A23 stated that the allocation of r_1 does not need to be considered, when allocating r_2 and r_3, both concluded differently on how r_2 and r_3 should be allocated: supporters of A21 argued that it is fair to *allocate the resources fair*, *i.e.*, give both an equal amount of the resources, while supporters of A23 argued that it is fair to *allocate the deficit fair*, *i.e.*, give each consumer one resource less, although this means that c_2 receives two more additional. Allocating the deficit in a fair manner, was frequently justified by the assumption that demands reflect actual needs. In general, participants considered the information important, which consumer ceased more demands for r_1 and whether demands reflect actual needs or negotiation positions. Although A22 is most fair according to all metrics discussed in Section 3.2.3.1, only 23.7% of participants selected it. In contrast, A23 was selected by 62.4% of participants, that is, almost three times as often. This shows that most participants consider it fair to allocate the deficit fairly and not the actual resources.

Unfortunately, giving a larger amount of resources to consumers, who demand more, gives incentives to cloud users to exaggerate demands, which is already sometimes the case [30]. Therefore, this result, although interesting, cannot be applied to allocate cloud resources, as it would allow for strategic manipulation and potentially give incentive to waste resources.

3.2.4.3 Estimating Greediness (Q3)

Many participants stated to have struggled ranking the consumers, because no information was given about how efficiently these consumers used their

resources. Unfortunately, such uncertainties are also present in clouds, where the performance of VMs can only in some cases be related to their resource consumption.

Numbers of Table 3.4 show that these rankings according to the four metrics presented in Section 3.2.3.1 cover 90.7% (S31), 94.8% (S32), and 73.6% (S33) of the participants' rankings. Despite this high coverage it was often the case that a participant's ranking was conforming with different metrics over the three scenarios. This indicates that participants have an intuitive understanding of greediness, which they try to justify proposing a metric, although the metric does not work for all scenarios to produce the ranking they give. The four most prominent combinations ((2,1,3), (2,3,1), (2,3,1)), ((1,2,3), (1,3,2), (2,3,1)), ((1,2,3), (1,3,2), (1,3,2)), and ((2,1,3), (3,2,1), (1,2,3)) are conforming with DoRe, C×S, Cost, C∩S, respectively. However, as discussed next, the two top combinations often did not result from applying the respective metric.

The most prominent combination was chosen by 79 participants. Although this combination results from the DoRe metric, only one participant argued according to this metric. Interestingly, participants who opted for this combination provided more often explanations than those who opted for other combinations. The most frequent argument was, that those who exceed their equal share of a resource are considered greedy. Thus, in S31 c_2 is the greediest due to the disproportional consumption of r_3. Analog, in S32 and S33 c_3 is the greediest, because c_3 exceeds the equal share of r_2 by 50% and c_1 is the second greedy, because c_1 exceeds the equal share of r_1 by 33%. The combination of rankings chosen by 55 participants is conforming with the C×S metric. Applying this metric implies several calculations, but participants arrived at this ordering differently, as inferred from their comments: most ordered the consumers according to the overall amount they had used and broke the tie of c_1 and c_3 in S33 by c_3's 50% overconsumption of r_2. The combinations of rankings chosen by 43 and 32 participants are conforming with the cost and C∩S metric, respectively. Most participants stated that they had applied these metrics to arrive at the ranking. Note that, for the combination of the C∩S metric, the ranking from S32 to S33 is inverted, where in S32 c_1 was least greedy and c_3 most greedy. Two participants already

noted in their comments that this reordering is counter-intuitive, because c_1 actually takes up less resources in S33, while consumptions of the other two consumers are stable. Based on these and the results of Section 3.2.4.1 the following conclusions on the sufficiency of those metrics are drawn.

3.2.4.4 Implications for Existing Metrics

Although the metrics discussed in Section 3.2.3.1 cover the majority of participants rankings, none of these metrics captures an intuitive understanding of fairness: The C∩S metric has a low conformance in S32 and S33 and for S33 results in the inverted ranking of S32, which means that consumers can decrease their ranking by adding additional resources to their bundle. This not only is counter-intuitive, but also gives incentives to consumers to consume more resources than needed. The Cost metric has an insufficient conformance in S33 and identifies c_1 as "greediest" in S31, although c_1 does not cause the bottleneck, but precisely sticks to his equal share (a behavior that is considered humble in S32 and S33). The sum- and the root-based-penalty function presented in [48] result in the same rankings as the Cost metric and, thus, are also not satisfactory. Similar arguments apply to the C×S metric.

The DoRe metric is satisfactory at a first glance: For all three scenarios of Q3 it results in the most frequent ranking and also in the most frequent combination of rankings (cf. first body row of Table 3.4 and Table 3.5). However, only one participant argued according to the DoRe metric, while the majority argued that those who exceed their equal share are greedy. Therefore, the high conformance of the DoRe metric stems rather from the fact that every consumer exceeds his equal share on at most one resource, which allows the DoRe metric to produce good results, because it only considers these resources. To show that DoRe metric's approach to ignore all but one resource can lead to undesirable results, a sample allocation is presented in Table 3.6. For this allocation all other metrics discussed in Section 3.2.3.1 give the inverse ranking of the DoRe metric. Also, according to the arguments made by the participants, the DoRe ranking is unfair: The DoRe metric classifies c_1 as the consumer with the most valuable bundle, although c_1 only receives the least loaded resource. c_3 cedes no resource at all and receives most of the only scarce resource, but the DoRe metric classifies c_3 as most humble.

Table 3.6: An example of a problematic DoRe metric ranking

	r_1	r_2	r_3	Cost	C×S	C∩S	DoRe
available	30	30	30				
c_1	18	0	0	1.80	1.68	0.00	0.60
c_2	0	14	17	3.10	3.04	1.40	0.56
c_3	10	16	12	3.80	3.69	1.60	0.53

Q1 of the questionnaire identified A13 as the intuitively fair allocation, while the DRF allocation (therefore, also most fair according to the DoRe metric) is A12. Because Leontief utility functions are assumed in Q1, *i.e.*, resources are required in static ratios, and DRF is defined based on this assumption, DRF should result in an intuitively fair allocation, when this assumption holds. Therefore, while DRF is often applied, when Leontief utility functions do not hold, Q1 shows that already for Leontief utility functions, DRF may result in allocations that are not intuitively fair. Moreover, consumers and resources can be added to this scenario, where consumer c_i requests only resource r_i (and as before one consumer requests all resources evenly). Thereby, the perceived unfairness of DRF can be increased arbitrarily, because the consumer requesting all resources would receive as much as all other consumers combined (cf. Section 2.6.1.4.2). In other words, DRF allocations exist with a higher degree of intuitive unfairness.

3.3 Approach

This section takes the following steps in order to define cloud fairness. The insights gathered in Section 3.2 about an intuitive understanding of greediness are deployed in Section 3.3.1 to develop a novel bundle measure termed the Greediness Metric (GM). GM quantifies the greediness of consumers based on their multi-resource self-servings. Therefore, when consumers with unknown utility functions serve themselves from a free-for-all resource pool, it is fair to constrain consumers in proportion to their GM value in favor of other consumers. Section 3.3.2 develops a formal cloud model and formulates incentives that have to be provided to users. Section 3.3.3 refines GM

such that it is applicable to this cloud model and Section 3.3.4 proves that enforcing fairness based on this refined GM provides the designated incentives.

3.3.1 Greediness Metric

Due to the shortcomings of all existing metrics presented in Section 3.2.4.4 and their inability to capture an intuitive understanding of greediness, a new Greediness Metric (GM) is developed. This GM is aligned with the arguments of participants and accordingly (a) results in the most frequent rankings for each of the three scenarios in Q3, (b) classifies A13 as the most fair allocation, and (c) gives the "correct" ranking for the allocation in Table 3.6.

GM is a bundle measure and, therefore, maps each resource bundle in an allocation to a rational number that can be associated to the *greediness* of the consumer, who served herself the bundle. The GM sums up, what exceeds the equal share in each bundle, according to the most frequent questionnaire's responses. However, it also deducts what is not consumed of the equal share and is handed over to other consumers instead.

3.3.1.1 GM Definition

Let matrix $A \in \mathbb{R}_{\geq 0}^{\mathfrak{r} \times \mathfrak{c}}$ describe an allocation of $\mathfrak{r}$ resources to $\mathfrak{c}$ consumers, as discussed in Section 2.1. The amount of r_i that c_j receives beyond his equal share is then $a_{ij} - \overleftrightarrow{r_i}/\mathfrak{c}$ (if the difference is negative, c_j does not utilize its entire equal share of the resource).

If $a_{ij} > \overleftrightarrow{r_i}/\mathfrak{c}$, consumers other than c_j have to cede some of their equal share of r_i in order to enable c_j's share of r_i. Therefore, the amount that exceeds c_j's equal share is added to the greediness of c_j.

If $a_{ij} = \overleftrightarrow{r_i}/\mathfrak{c}$, c_j exactly receives his equal share, wherefore it does not change c_j's greediness. In particular, if $a_{ij} = \overleftrightarrow{r_i}/\mathfrak{u}$ for all $i \in \{1, 2, \ldots, \mathfrak{r}\}$, c_j's greediness is zero.

If $a_{ij} < \overleftrightarrow{r_i}/\mathfrak{c}$, c_j's cession of r_i is credited to c_j, *i.e.*, subtracted from c_j's greediness, to the extent that other consumers profit from this cession, which is the case, when they utilize r_i beyond their equal share. This extent not only depends on how much of r_i is utilized beyond the equal share by other consumers, but also on how much of r_i is ceded by other consumers. Therefore, the *credit factor* for the cession of r_i is the ratio of what is ceded of r_i to what

is consumed beyond the equal share of r_i. To capture this notion formally, $\mathsf{b}(r_i)$ defines the sum of what consumers receive beyond their equal share of r_i, *i.e.*,

$$\mathsf{b}(r_i) := \sum_{j=1}^{\mathfrak{c}} \max\left(0, a_{ij} - \frac{\overleftrightarrow{r_i}}{\mathfrak{c}}\right),$$

and $\mathsf{c}(r_i)$ defines the sum of what consumers cede of r_i, *i.e.*,

$$\mathsf{c}(r_i) := \sum_{j=1}^{\mathfrak{c}} \max\left(0, \frac{\overleftrightarrow{r_i}}{\mathfrak{c}} - a_{ij}\right).$$

Multiplying the amount that c_j cedes of r_i with $\mathsf{b}(r_i)/\mathsf{c}(r_i)$ implements the considerations above. Therefore, the *greediness of* c_j is defined as follows.

Definition 3.1. The *greediness of consumer* $c_j \in C$ is defined by

$$\mathsf{G}(c_j) := \sum_{i=1}^{\mathfrak{r}} \frac{\mathfrak{c}}{\mathfrak{r} \cdot \overleftrightarrow{r_i}} \cdot \left(a_{ij} - \frac{\overleftrightarrow{r_i}}{\mathfrak{c}}\right) \cdot \begin{cases} \mathscr{A} & \text{if} \quad a_{ij} \geq \frac{\overleftrightarrow{r_i}}{\mathfrak{c}}, \\ \mathscr{B} \cdot \frac{\sum_{j=1}^{\mathfrak{c}} \max\left(0, a_{ij} - \frac{\overleftrightarrow{r_i}}{\mathfrak{c}}\right)}{\sum_{j=1}^{\mathfrak{c}} \max\left(0, \frac{\overleftrightarrow{r_i}}{\mathfrak{c}} - a_{ij}\right)} & \text{else.} \end{cases} \tag{3.1}$$

The factor $\frac{\mathfrak{c}}{\mathfrak{r} \cdot \overleftrightarrow{r_i}}$ normalizes resource units. Section 3.3.1.3 discusses the normalization in detail. Note that, if $\mathsf{c}(r_i) = 0$, no consumer cedes r_i and, therefore, the else-part of Equation 3.1 is never reached (and no division by zero occurs). While $\frac{\mathsf{b}(r_i)}{\mathsf{c}(r_i)}$ in the else-part dynamically influences in dependence of the consumption of r_i how much ceding r_i is credited, this crediting is also statically adjusted by Parameter $0 \leq \mathscr{B} \leq 1$. In return, $1 \leq \mathscr{A}$ statically influences how much the exceeding of a resources is credited. Section 3.3.4.4 determines a range of concrete values for $\mathscr{A}$ and $\mathscr{B}$ based on the questionnaire results. In particular, when $\mathscr{A}$ and $\mathscr{B}$ are determined according to Equation 3.20, whereat $0 \leq \mathscr{D} \leq 1$, GM perfectly conforms with the questionnaire results.

3.3.1.2 Examples

The complexity of the greediness metric is higher than of those metrics presented in Section 3.2.3.1. In order to help understanding how the different

components of GM play together, this section exemplarily calculates GM for selected allocations of Q1 and Q3 of the questionnaire.

3.3.1.2.1 A12

In the scenario of Q1, one unit of r_1 has a normalized value of $\mathfrak{c}/(\mathfrak{r} \cdot \overleftrightarrow{r_1}) = 3/(2 \cdot 6) = 0.25$ and one unit of r_2 has a normalized value of $\mathfrak{c}/(\mathfrak{r} \cdot \overleftrightarrow{r_2}) = 3/(2 \cdot 12) = 0.125$. The equal shares are $\overleftrightarrow{r_1}/3 = 6/3 = 2$ and $\overleftrightarrow{r_2}/3 = 12/3 = 4$ for resource r_1 and r_2, respectively. c_1 has a greediness of

$$\mathsf{G}(c_1) = \overbrace{0.25 \cdot (3-2) \cdot \mathscr{A}}^{\text{Overcons. of } r_1} + \overbrace{0.125 \cdot (0-4) \cdot \frac{2+2}{4} \cdot \mathscr{B}}^{\text{Ceding of } r_2} = 0.25 \cdot \mathscr{A} - 0.5 \cdot \mathscr{B}.$$

In particular, the fraction in the second summand results from c_1 ceding four units of r_2 and c_2 and c_3 exceeding their equal share of r_2 by two units each. The subtraction in brackets results because c_1 consumes zero units of r_2, while the equal share is 4.

For c_2 the calculations are inverse, with respect to r_1 and r_2, *i.e.*,

$$\mathsf{G}(c_2) = \overbrace{0.25 \cdot (0-2) \cdot \frac{1+1}{2} \cdot \mathscr{B}}^{\text{Ceding of } r_1} + \overbrace{0.125 \cdot (6-4) \cdot \mathscr{A}}^{\text{Overcons. of } r_2} = 0.25 \cdot \mathscr{A} - 0.5 \cdot \mathscr{B}.$$

Because c_3 does not cede resources, c_3's greediness does not depend on $\mathscr{B}$:

$$\mathsf{G}(c_3) = \overbrace{0.25 \cdot (3-2) \cdot \mathscr{A}}^{\text{Overcons. of } r_1} + \overbrace{0.125 \cdot (6-4) \cdot \mathscr{A}}^{\text{Overcons. of } r_2} = 0.5 \cdot \mathscr{A}.$$

3.3.1.2.2 S32 and S33

S32 and S33 of Q3 are similar: the amounts of available resources are the same, resulting in both scenarios in normalized values of $0.08\overline{3}$, $0.08\overline{3}$, and $0.\overline{1}$ and equal shares 4, 4, and 3 for r_1, r_2, and r_3, respectively. Also the allocations are the same, except that c_1 cedes one unit more of r_2 in S33 compared to S32. Because this unit is not utilized by other consumers, this ceding is not credited to c_1, *i.e.*, c_1's greediness does not change. In particular, the greediness

of c_1 is calculated as follows:

$$G(c_1) = \overbrace{0.08\bar{3}\cdot(4-4)\cdot\mathscr{A}}^{\text{Cons. of } r_1} + \overbrace{0.08\bar{3}\cdot(-2\cdot\mathscr{B})}^{\text{Cons. } r_2} + \overbrace{0.\bar{1}\cdot(4-3)\cdot\mathscr{A}}^{\text{Cons. of } r_3} = 0.\bar{1} - 0.1\bar{6}\cdot\mathscr{B},$$

Factor $(-2\cdot\mathscr{B})$ in the Consumption of r_2 is calculated by

$$(2-4)\cdot\frac{2}{2}\cdot\mathscr{B} = -2\cdot\mathscr{B}$$

in S32 and in S33 by

$$(1-4)\cdot\frac{2}{3}\cdot\mathscr{B} = -2\cdot\mathscr{B}.$$

3.3.1.3 Resource Normalization

Because different resources occur in different amounts, it is necessary to normalize the offsets before summation (cf. Equation 3.1). To do so, a normalization constant $k > 0$ needs to be fixed, such that $k/\overleftrightarrow{r_i}$ can be used as normalization factor for every $r_i \in R$. This section motivates why GM defines $k = \mathfrak{c}/\mathfrak{r}$, *i.e.*, that k increases proportionally with the number of consumers and decreases proportionally with the number of resources.

3.3.1.3.1 Increase with the Number of Consumers

The addition of a consumer is neutral, when it does not change the overall availability of resources. Therefore, the neutral addition of a consumer must not change the greediness of consumers that were already there. This implies increasing k proportionally with the number of consumers, as shown subsequently.

Let A be an allocation of $\mathfrak{r}$ resources to the consumers $c_1, \ldots, c_\mathfrak{c}$ and let $es \in \mathbb{R}^\mathfrak{r}_{>0}$ be the equal share. Add x times es resources and add x consumers of which each precisely consumes es resources. Then these x added consumers are neutral. To ensure that the greediness of $c_1, \ldots, c_\mathfrak{c}$ does not change, k must increase proportionally to the number of consumers, *e.g.*, $k = \mathfrak{c}$ before the x consumers are added and $k = \mathfrak{c} + x$ thereafter (if k would be con-

stant, the greediness of consumers would decrease, when neutrally adding consumers).

3.3.1.3.2 Decrease with the Number of Resources

The addition of a resource is neutral, when it does not change the relative overall allocation of consumers. Therefore, the neutral addition of a resource must not change the greediness of consumers. This implies decreasing k proportionally with the number of resources, as shown subsequently.

Let $\mathfrak{r} = 1$, *i.e.*, there is only resource r_1 of which a certain amount is available. Let A^1 be an allocation of r_1 to $\mathfrak{c}$ consumers. Add y resources and allocate each of these in the same ratio as A^1 allocates r_1. Call this allocation A^y. Then, proportionally A^1 and A^y allocate all consumers the same share of overall resources. Therefore, the addition of the y resources is neutral. To ensure that the greediness of consumers is the same for A^1 and A^y, k must decrease proportionally with the number of resources, *e.g.*, $k = 1$ before the y resources are added and $k = \frac{1}{y+1}$ after the y resources are added (if k would be constant, the greediness of consumers would increase by factor y).

3.3.2 Cloud Model

GM, as defined above, is not applicable to clouds for the following reasons. GM defines the greediness of consumers, who share the same self-serving store. Therefore, GM can be applied to VMs that share the same node. However, fairness is not to be enforced among VMs but among users and users operate VMs on different nodes. Furthermore, even if fairness would have to be enforced among VMs, it would have to be accounted for the VR configuration of VMs. In particular, VMs may vary significantly in the amount of their VRs and, therefore, a VM with more VRs, should be allowed to utilize more PRs before being identified as greedy and being constrained. Lastly, prioritizing less greedy users should provide incentive users to configure the VRs of their VMs correctly, as correctly configured VMs can be scheduled more efficiently.

In order to deepen this discussion and allow for the refinement of GM such that it is applicable to clouds, a formal and exhaustive model of cloud multi-resource allocation is required. Because controlling cloud resources

through PR allocation was a relatively unexplored research field before this thesis, such model did not exist and is developed in Section 3.3.2.1. Section 3.3.2.2 formulates three incentives that should provided, when enforcing fairness in clouds and Section 3.3.2.3 develops a technical definition essential for these incentives. Lastly, Section 3.3.2.4 differentiates two resource scales.

3.3.2.1 Cloud Representation

Formally a cloud is represented by a set of users $U = \{u_1, u_2, \ldots, u_{\mathfrak{u}}\}$, a set of nodes $N = \{n_1, n_2, \ldots, n_{\mathfrak{n}}\}$, and a set of VMs $V = \{v_1, v_2, \ldots, v_{\mathfrak{v}}\}$. A VM is owned by a user and hosted by a node, whereat the cloud scheduling policy (and not the user) decides which node hosts the VM. Function $\mathsf{own}\colon V \cup U \to U \cup \mathcal{P}(V)$ maps (a) a VM to its owner and (b) a user to the set of VMs the user owns. Function $\mathsf{host}\colon V \cup N \to N \cup \mathcal{P}(V)$ maps (a) a VM to the node that hosts the VM and (b) a node to the set of VMs the node hosts. Function $\mathsf{nbr} : V \to \mathcal{P}(V)$ maps $v \in V$ to the set of "neighboring" VMs, *i.e.*, VMs that run on v's host, or, formally $\mathsf{nbr}\colon v \mapsto \{\dot{v} \in V | \mathsf{host}(\dot{v}) = \mathsf{host}(v)\}$. Users and VMs are also referred to as entities.

VMs share their host's PRs such as CPU time, RAM, disk I/O, and network access. Let $R = \{r_1, r_2, \ldots, r_{\mathfrak{r}}\}$ be the set of PRs to be considered for a fair allocation. VMs are defined by Virtual Resources (VRs), *e.g.*, virtual CPU (VCPU) and virtual RAM (VRAM), often chosen from a range of different *flavors*, *i.e.*, a VM flavor is a set of VRs that a VM of that flavor has. For now, it is assumed, that for every PR a virtual counterpart exists, although this is not necessarily the case. However, a nonexistence can be treated easily by defining such values based on VRs that exist. Especially in private clouds, PRs may be managed by quotas, *i.e.*, each user has a quota that defines a maximum of VRs that the user's VMs may have in total, which are denoted by function $\mathsf{quota}\colon U \to \mathbb{R}^{\mathfrak{r}}_{\geq 0}$. Function $\mathsf{phy}\colon N \to \mathbb{R}^{\mathfrak{r}}_{\geq 0}$ maps nodes to their PRs. The information about $\mathsf{phy}(n)$ is also referred to as node n's Node Resource Information (NRI) and the Cloud Resource Supply (CRS) is the sum of NRI of all nodes in the cloud, *i.e.*,

$$\mathsf{crs} := \sum_{n_j \in N} \mathsf{phy}(n_j). \tag{3.2}$$

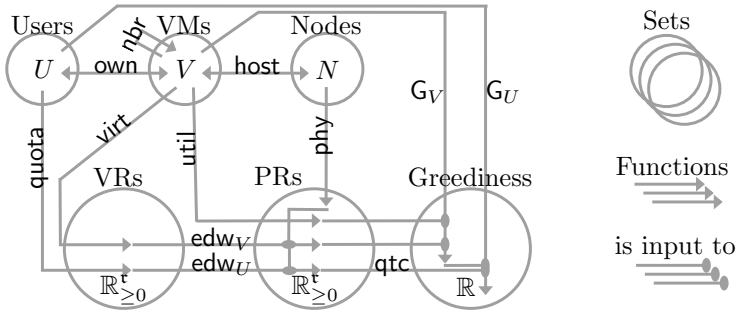


Figure 3.14: Dependencies among the definitions of Chapter 3

Function $\mathsf{virt}\colon V \to \mathbb{R}^{\mathfrak{r}}_{\geq 0}$ maps VMs to their VRs and function $\mathsf{util}\colon V \to \mathbb{R}^{\mathfrak{r}}_{\geq 0}$ maps VMs to the PRs they utilize, *i.e.*, the load they impose on the PRs (at a distinct point in time). Arithmetic operations on vectors are applied point-wise.

Figure 3.14 illustrates the definitions required. The upper three circles represent the three sets U, V, and N, as explained above. The lower three circles represent the VRs, PRs, and greediness codomain. As there is more than one VR and more than one PR, the former two sets are vector spaces, while the greediness codomain is the set of real numbers. Functions are represented by arrows between the sets, *e.g.*, virt pointing from V to the VRs circle depicts that function virt maps VMs to their VRs. The endowment function edw_V (cf. Definition 3.2) determines a VM's endowment to PRs based on the VM's VRs. Therefore, edw_V connects the virt-arrow to the PRs circle. However, edw_V also depends on the PRs of the VM's host. Therefore, the end of phy is connected to edw_V. Note that, although the edw_V-arrow begins in the VRs circle, function edw_V actually maps from set V. G_V maps a VM to its greediness based in the VM's PR utilization (util) and the VM's endowment (edw_V). Therefore, G_V connects Users to Greediness and is connected to by the ends of util and edw_V.

3.3.2.2 Incentives

A cloud budgets a certain amount of PRs for a VM based on the VM's VRs. Therefore, during VM scheduling, a node to host the VM will be selected that can best serve the budgeted PRs. For example, placing "small" VMs on

nodes with less remaining capacity increases the utilization of these nodes and leaves nodes with more remaining capacity free to accommodate "large" VMs that cannot be hosted by nodes with less remaining capacity. Accordingly, a VM with a PR utilization that strongly deviates from what is budgeted based on the VM's VRs, leads to either over-loaded or under-utilized PRs on the VM's host, *i.e.*, higher stress for the cloud. Therefore, enforcing fairness must give incentive to users to configure their VMs properly, *i.e.*, their VMs must have the highest priorities, when the VMs' configurations perfectly match their PR utilization. This is referred to as the *configuration incentive*.

However, users are not always able to precisely predict the PR utilization of their VMs or the PR utilization will be fluctuant. Therefore, when users are uncertain about the upcoming PR utilization of VMs, enforcing fairness must make it advantageous for users to configure VMs to the best of their knowledge. This is referred to as the *uncertainty incentive*.

Although it can be argued that several small VMs are easier to place on nodes than one large VM (with the same sum of VRs), several small VMs imply higher PR usage, as every VM runs its own OS. Also some users may be able to partition their workloads, while others are not. Therefore, enforcing fairness must neither give incentive to partition VRs and workloads to multiple small VMs nor to concentrate VRs and workloads to one monolithic VM. Therefore, how a user partitions a fixed amount of VRs to VMs must not influence the prioritization of the user's VMs, when the VMs are idle. This is referred to as the *partition incentive* and is only demanded for idle VMs, as the configuration incentive demands to relate a VM's configuration to the VM's PR utilization and collides with the partition incentive, if VMs are not idle.

3.3.2.3 VM Endowments

The configuration incentive demands that configuring VMs in conformance with the subsequent PR utilization is rewarded, when fairness is enforced. The rationale behind this is that based on a VM's VRs the cloud expects a certain load of the VM and schedules the VM on a node that can most economically provide for this anticipated load. In particular, depending on the cloud's overcommit ratios and a VM's VRs, the cloud budgets a certain amount of

PRs for the VM. Accordingly, a VM's *endowment* is defined as the amount of PRs that are budgeted for this VM. This definition allows for multiple functions $\mathsf{edw}_V \colon V \to \mathbb{R}^{\mathsf{r}}_{\geq 0}$ to calculate the endowment. Subsequently, three reasonable functions are discussed.

The straight-forward endowment function is to scale a VM's VRs by the ratio of the CRS to the number VRs that can maximally be assigned to VMs, *i.e.*,

$$\mathsf{edw}_V(v_i) \mapsto \frac{\mathsf{crs}}{\sum_{u_k \in U} \mathsf{quota}(u_k)} \cdot \mathsf{virt}(v_i). \tag{3.3}$$

However, this formula does not account for the number of VMs that are actually instantiated. This can be overcome by defining the endowment function to scale a VM's VRs by the ratio of the CRS to the VRs of running VMs, *i.e.*,

$$\mathsf{edw}_V(v_i) \mapsto \frac{\mathsf{crs}}{\sum_{v_k \in V} \mathsf{virt}(v_k)} \cdot \mathsf{virt}(v_i). \tag{3.4}$$

However, depending on the ratio of VRs the VM will be scheduled to different nodes, wherefore the VM's host's PRs are the best indicator of how many PRs are budgeted for a VM. Accordingly, the endowment of a VM is defined as follows.

Definition 3.2. The *endowment* of a VM v_i is defined as v_i's proportional share of v_i's host's PRs and denoted by

$$\mathsf{edw}_V(v_i) \mapsto \min\left(\mathsf{virt}(v_i), \frac{\mathsf{phy}(\mathsf{host}(v_i))}{\sum_{v_j \in \mathsf{nbr}(v_i)} \mathsf{virt}(v_j)} \cdot \mathsf{virt}(v_i)\right). \tag{3.5}$$

Definition 3.2 defines the endowment of a VM as the share of the VM's host's PRs that is proportional to the VM's VRs. By taking the point-wise minimum, it is ensured that a VM's endowment is not greater than the VM's VRs. This is desirable, as a VM can be scheduled to a node, that is currently not "saturated" with VMs. In this case and without the minimization, the VM's endowment would exceed the VM's VRs. As a VM's VRs specify the capacity that the VM's owner planned for the VM, the cloud will never budget more PRs for a VM than the VM's VR, wherefore the minimization is necessary.

3.3.2.4 Overcommitment and Resource Scales

Overcommitting PRs (cf. Section 2.2.1.5) allows the sum of quotas to exceed the CRS and the sum of VRs of VMs hosted by a node to exceed the node's PRs. In particular, the factor by which quotas and VRs exceed the CRS and PRs is defined by the overcommit ratios. Therefore, quotas and VRs are on the *virtual scale* and PRs are on the *actual scale*. As Definition 3.2 maps a VM's VRs to a proportion of the PRs, endowments are also on the actual scale. In particular, calculating the endowment can be considered mapping VRs to the actual scale.

3.3.3 Cloud Greediness and Fairness

This Section defines VM greediness and user greediness and based on these definitions cloud fairness. It is assumed that PR amounts are normalized according to the CRS.

3.3.3.1 VM Greediness

Definition 3.3 refines Definition 3.1 for VMs and is parameterized by δ to account for different cloud utilization levels (cf. Section 3.3.4.3 and 3.3.4.4).

Definition 3.3. For $0 \leq \delta \leq 1$, function $\mathsf{G}_V^\delta \colon V \to \mathbb{R}$ defines the *VM greediness* of VM v_i by

$$\mathsf{G}_V^\delta(v_i) := \sum_{j=1}^{r} \mathsf{edw}_V(v_i)_j + (\mathsf{util}(v_i)_j - \mathsf{edw}_V(v_i)_j) \cdot \frac{2}{\delta + 2} \cdot \begin{cases} 2 & \text{if } \mathsf{util}(v_i)_j \geq \mathsf{edw}_V(v_i)_j, \\ \overbrace{\min\left(1, \frac{\sum_{v_k \in \mathsf{nbr}(v_i)} \max\left(0, \mathsf{util}(v_k)_j - \mathsf{edw}_V(v_k)_j\right)}{\sum_{v_k \in \mathsf{nbr}(v_i)} \max\left(0, \mathsf{edw}_V(v_k)_j - \mathsf{util}(v_k)_j\right)}\right)}^{\text{Dynamic Ceding Factor (DCF)}} & \text{else.} \end{cases} \tag{3.6}$$

The greediness of VM v_i depends on $\mathsf{util}(v_i)$ and $\mathsf{edw}_V(v_i)$, and these two parameters of the VMs that run *on the same node* as v_i. Thus, the

VM greediness is a sum of values on the actual scale. The first summand $(\sum_{j=1}^{r} \mathsf{edw}_V(v_i)_j)$ is called the *static greediness* of v_i and the second summand the *dynamic greediness* of v_i.

Consumers in Definition 3.1 are equal and, therefore, have the same equal shares. In contrast, VMs have very different VRs and, therefore, endowments, which represent their entitlements to the host's PRs. Definition 3.3 takes these differences into account and "rewards" correct VM configuration by a lower greediness (cf. Section 3.3.4.2). This leads to the following differences of Equation 3.6 and Equation 3.1.

1. The first summand $(\mathsf{edw}_V(v_i)_j)$ inside the sum function of Equation 3.6 has no equivalent in Equation 3.1 and sums up v_i's endowment (cf. Definition 3.2). This summand is necessary to ensure (a) that the greediness of a VM is never negative and, therefore, instantiating a VM never decreases a user's greediness and (b) that the "cost" of instantiating a VM increases with the VM's VRs. Section 3.3.4.1 proves that this allows providing the partition incentive.

2. Equation 3.1 uses the equal share $(\overleftrightarrow{r_i}/u)$ and Equation 3.6 uses the endowment as reference point to calculate the deviation from the appropriate PR utilization. Section 3.3.4.2 proves that this allows to provide the configuration incentive.

3. Equation 3.6 ensures that the Dynamic Ceding Factor is not greater 1, and, therefore, ceding a PR is not "rewarded" stronger than what is actually ceded of the PR. This is necessary for the case in which VMs are scheduled to a node that is not yet "saturated" with VMs, *i.e.*, the sum of VRs of VMs hosted by the node is smaller than the node's PRs. This minimization is not necessary in Equation 3.1, as endowments are defined by the equal share and, therefore, the sum of endowments equals the overall supply.

4. Equation 3.1 uses the factors $\mathscr{A}$ and $\mathscr{B}$, while Equation 3.6 replaces these factors by $\frac{4}{\delta+2}$ and $\frac{2}{\delta+2}$, respectively, for $0 \leq \delta \leq 1$. This change allows to provide the uncertainty incentive and full compliance with the questionnaire results, as discussed in Section 3.3.4.3 and 3.3.4.4.

3.3.3.2 User Greediness

The greediness of a user u_i is mainly determined by sum of greediness of u_i's VMs. However, users can be heterogenous in terms of their entitlements to the cloud's resources. In particular, depending on the importance of users or other differentiation criteria, users can have different quotas. Let two users $u_1, u_2 \in U$ instantiate an identical VM (in terms of VRs and actual PR utilization) and let u_1 have a larger quota than u_2. Then, the greediness of u_2 must be larger, since u_2 imposes the same stress on the cloud as u_1 but has a smaller entitlement to the cloud's resources. Therefore, a user's greediness must decrease with the user's quota.

A user u_i's greediness contains the sum of greediness of u_i's VMs and therefore a sum of values on the actual scale. Thus, u_i's quota has to be mapped to the actual scale, before factoring it into the u_i's greediness. This mapping is done by the user endowment.

Definition 3.4. The *user endowment* of a user u_i is defined as the share of the cloud's PRs that is proportional to u_i's quota and calculated by function

$$\mathsf{edw}_U(u_i) \mapsto \frac{\mathsf{crs}}{\sum_{u_k \in U} \mathsf{quota}(u_k)} \cdot \mathsf{quota}(u_i). \tag{3.7}$$

The sum of values in this vector is u_i's *quota credit*, which is defined by function

$$\mathsf{qtc} := \sum_{j=1}^{\mathfrak{r}} \mathsf{edw}_U(u_i)_j. \tag{3.8}$$

u_i's greediness is defined by the sum of u_i's VMs' greediness subtracted by u_i's quota credit.

Definition 3.5. The *δ-user greediness* of user $u_i \in U$ is defined by

$$\mathsf{G}^{\delta}_U(u_i) := \left(\sum_{v_j \in \mathsf{own}(u_i)} \mathsf{G}^{\delta}_V(v_j) \right) - \mathsf{qtc}(u_i). \tag{3.9}$$

3.3.3.3 Cloud Fairness

Definition 3.6 provides a highly practical definition of cloud fairness based on the greediness metrics developed above. The definition accounts for the constraints identified in Section 3.1 by avoiding assumptions on utility functions and instead defining fairness via a prioritization scheme based on the greediness of users.

Definition 3.6. *Cloud fairness* is the procedure of prioritizing VMs inversely to the greediness of their users which is defined by Definition 3.5.

Definition 3.6 has the following three advantages:

1. Assumptions on utility functions are avoided by only taking into account (a) what the users' entitlement to the cloud's resources are (based on their quotas), (b) which PRs VMs utilize, and (c) how VMs are configured in terms of VRs.

2. The definition accounts for the fact that the allocation of most PRs is managed by proportional priorities (cf. Section 2.2.1.1), and, therefore, the amount a VM receives of a PR cannot be configured but instead proportional priorities have to be assigned, such that the designated allocation is approximated.

3. As Definition 3.1 captures an intuitive understanding of greediness and Definition 3.5 is based upon this definition, it is intuitively fair to constrain greedy users in favor of less greedy users. Thus Definition 3.6 complies with an intuitive understanding of fairness.

Subsequently, Definition 3.6 is generalized to prove that the special case it constitutes is most advantageous. In particular, the generalizations allow defining greediness by various metrics and enforcing fairness on a per-user or per-VM basis.

Definition 3.6 consist of two parts: (a) the procedure of prioritizing VMs inversely to the greediness of their owners and (b) defining the owners' greediness by Definition 3.5. Definition 3.7 generalizes Definition 3.6 by relaxing the latter part and allowing to define greediness by various metrics.

Definition 3.7 is subsequently deployed to show that prioritizing VMs inversely to the greediness of their owners exhibits several desirable characteristics, when greediness is defined by Definition 3.5 but not when defined by common norms such as the L_∞ or L_1 norm.

Definition 3.7. For metric M, the M_g*-policy* is the procedure of prioritizing VMs inversely to the greediness of their users which is defined by metric M.

Therefore, the G^{δ}_{Ug}-policy enforces cloud fairness as defined by Definition 3.6. Lastly, prioritizing VMs inversely to their own greediness is defined and will be evaluated subsequently as well. Definition 3.7 and 3.8 are distinguished by the *g* and *l* subscript.

Definition 3.8. For metric M, the M_l*-policy* is the procedure of prioritizing VMs inversely to their greediness as defined by metric M.

3.3.3.4 Prioritization and Max-min Fairness

VM demands constantly change during runtime. In order to be applicable in such dynamic environment and because PRs during VM runtime are controlled by proportional priorities, fairness in Section 3.3.3.3 is formulated via prioritization schemes. This raises the question of how a fair allocation looks like according to Definitions 3.6, 3.7, and 3.8, *i.e.*, how does an allocation look like that does not change when calculating priorities according to these definitions and allocating more PRs to VMs with higher priorities? Theorem 3.9 and Corollary 3.10 answer this question. It is assumed that VM demands are stable, as changing VM demands imply a changing allocation.

Theorem 3.9. *When VM demands are stable, an allocation A of the cloud's PRs to VMs that is* M*-max-min fair among users does not change when enforcing the* M_g*-policy. Therefore, A is fair according to the* M_g*-policy.*

Proof. Let *C* be a cloud as defined in Section 3.3.2, whereat VM demands are stable, and *A* be an allocation of *C*'s PRs to VMs, such that *A* is M_g-max-min fair among users (cf. Section 2.4.3). Take *A* as starting point and apply the M_g-policy as defined in Definition 3.7, *i.e.*, (a) calculate the greediness of users according to M_g, (b) assign priorities to VMs inversely the greediness

of their owner, (c) and move PRs from VMs with lower priorities to VMs with higher priorities.

Let $v_1, v_2 \in V$ be two VMs. To prove Theorem 3.9, it has to be shown that no PRs are moved between v_1 and v_2. When $\mathsf{host}(v_1) \neq \mathsf{host}(v_2)$ this follows immediately, as PRs cannot be moved between VMs that run on different hosts. Also when the VMs have the same priority, no PRs are moved between them. Thus, assume that $\mathsf{host}(v_1) = \mathsf{host}(v_2)$ and that

$$\mathsf{M}_g(\mathsf{own}(v_1)) > \mathsf{M}_g(\mathsf{own}(v_2)). \tag{3.10}$$

Then v_1 has a lower priority than v_2 and the enforcement of fairness according to Definition 3.6 demands moving PRs from v_1 to v_2. This increases $\mathsf{M}_g(\mathsf{own}(v_2))$. However, since A is M_g-max-min fair, $\mathsf{M}_g(\mathsf{own}(v_2))$ is already maximized by A. In particular, Equation 3.10 implies that at some point when generating A it was not possible to allocate more PRs to v_2, while v_1 could still be allocated PRs. This was either the case because v_2 was bottlenecked on a PR that v_1 did not request or v_2 already received all the PRs it requests. Therefore, it is impossible to move PRs from v_1 to v_2 and, thus, enforcing the M_g-policy does not move PRs between v_1 and v_2. Therefore, enforcing the M_g-policy does not change A and, thus, the M_g-max-min fair allocation A is also fair according to the M_g-policy. □

Theorem 3.9 implies that using the $L_{\infty g}$- and L_{1g}-policies enforce DRF and asset fairness, respectively. The following corollary can be proven by a slight adaption of the proof above.

Corollary 3.10. When VM demands are stable, an allocation A of the cloud's PRs to VMs that is M-max-min fair among VMs does not change when enforcing the M_l-policy. Therefore, A is fair according to the M_l-policy.

3.3.4 Incentives Provision and Questionnaire Compliance

This section proves that the incentives discussed in Section 3.3.2.2 are provided, when fairness is enforced according to Definition 3.6. Definition 3.6 provides these incentives, as it incorporates Definition 3.3. This section closes by showing that Definition 3.3 complies with the questionnaire results.

3.3.4.1 Partition Incentive

The partition incentive demands to neither provide incentive to users to artificially partition workloads to multiple small VMs nor to concentrate workloads to one monolithic VM. The following theorem shows that enforcing cloud fairness as defined by Definition 3.6 provides the partition incentive.

Theorem 3.11. *Enforcing cloud fairness according to Definition 3.6 provides the partition incentive.*

Proof. As Definition 3.6 demands prioritizing VMs of a user $u \in U$ based on u's greediness $\mathsf{G}_U(u)$, it has to be shown that $\mathsf{G}_U(u)$ is independent of how u partitions a fixed amount of VRs to idle VMs. Let $V_1, V_2 \subseteq V$ be two sets of idle VMs that have the same sum of VRs and run on the same node, *i.e.*, $\sum_{v_1 \in V_1} \mathsf{virt}(v_1) = \sum_{v_2 \in V_2} \mathsf{virt}(v_2)$ and $\forall v \in V_1 \cup V_2 \colon \mathsf{util}(v) = 0$. It has to be shown that the sum of greediness of both VM sets is equal.

Definition 3.2 implies $\sum_{v_1 \in V_1} \mathsf{edw}_V(v_1) = \sum_{v_2 \in V_2} \mathsf{edw}_V(v_2) = e$, for some $e \in \mathbb{R}^r$. From Definition 3.3 and $\forall v \in V_1 \cup V_2 \colon \mathsf{util}(v) = 0$ follows

$$\sum_{v_1 \in V_1} \mathsf{G}_V(v_1) = \sum_{v_1 \in V_1} \sum_{j=1}^{r} \mathsf{edw}_V(v_1)_j - \mathsf{edw}_V(v_1)_j \cdot \frac{2}{\delta + 2} \cdot \mathrm{DCF}_j \quad (3.11)$$

and

$$\sum_{v_2 \in V_2} \mathsf{G}_V(v_2) = \sum_{v_2 \in V_2} \sum_{j=1}^{r} \mathsf{edw}_V(v_2)_j - \mathsf{edw}_V(v_2)_j \cdot \frac{2}{\delta + 2} \cdot \mathrm{DCF}_j. \quad (3.12)$$

Equation 3.11 can be rewritten as

$$\begin{aligned} \sum_{v_1 \in V_1} \mathsf{G}_V(v_1) &= \sum_{v_1 \in V_1} \sum_{j=1}^{r} \mathsf{edw}_V(v_1)_j \cdot (1 - \frac{2}{\delta + 2} \cdot \mathrm{DCF}_j) && (3.13) \\ &= \sum_{j=1}^{r} \sum_{v_1 \in V_1} \mathsf{edw}_V(v_1)_j \cdot (1 - \frac{2}{\delta + 2} \cdot \mathrm{DCF}_j) && (3.14) \\ &= \sum_{j=1}^{r} e_j \cdot (1 - \frac{2}{\delta + 2} \cdot \mathrm{DCF}_j). && (3.15) \end{aligned}$$

Analogously, Equation 3.12 can be reformulated as

$$\sum_{v_2 \in V_2} G_V(v_2) = \sum_{j=1}^{r} e_j \cdot (1 - \frac{2}{\delta + 2} \cdot \mathrm{DCF}_j). \tag{3.16}$$

Thus, Equations 3.15 and 3.16 imply that $\sum_{v_1 \in V_1} G_V(v_1) = \sum_{v_2 \in V_2} G_V(v_2)$ and, therefore, the partition incentive is provided. □

3.3.4.2 Configuration Incentive

The configuration incentive demands that enforcing fairness must give incentive to users to configure their VMs such that the VMs' configurations perfectly match their PR utilization. The following theorem shows that the configuration incentive is provided, when cloud fairness as defined by Definition 3.6 is enforced.

Theorem 3.12. *Enforcing cloud fairness according to Definition 3.6 provides the configuration incentive.*

Proof. The theorem is proven on a per VM basis. In particular, as the priorities of VMs of a user are inverse to the user's greediness and the user's greediness includes the sum of greediness of the user's VMs, it has to be shown that the greediness of a VM is minimized, when the PRs it utilizes perfectly match its endowment.

Let $v_i \in V$ be a VM. Due to the first summand inside the sum function of Equation 3.6, v_i's greediness increases linearly with v_i's endowment. Let $r_j \in R$. If v_i's utilization of r_j exceeds v_i's endowment to r_j, v_i's greediness increases by the according difference scaled by $\frac{4}{\delta+2}$. That is, if $\mathrm{util}(v_i)_j > \mathrm{edw}_V(v_i)_j$, v_i's greediness increases by $\frac{4}{\delta+2} \cdot (\mathrm{util}(v_i)_j - \mathrm{edw}_V(v_i)_j)$. Because $\frac{4}{\delta+2} > 1$, for any $0 \leq \delta \leq 1$, v_i's greediness were less, if v_i's endowment to r_j were increased. However, increasing v_i's endowment to r_j such that it exceeds v_i's utilization of r_j is unfavorable: if v_i's endowment to r_j is greater than v_i's utilization of r_j, v_i's greediness decreases by the according difference scaled by $\frac{2}{\delta+2}$ and DCF. That is, if $\mathrm{edw}_V(v_i)_j > \mathrm{util}(v_i)_j$, v_i's greediness decreases by $\frac{2}{\delta+2} \cdot \mathrm{DCF} \cdot (\mathrm{edw}_V(v_i)_j) - \mathrm{util}(v_i)_j)$. Because $\frac{2}{\delta+2} \cdot \mathrm{DCF} \leq 1$, for any $0 \leq \delta \leq 1$, v_i's greediness were less (or at least not greater), if v_i's endowment

to r_j were decreased. It follows that $\mathsf{G}_V(v_i)$ is minimized, if v_i's PR utilization is equal to v_i's endowment, *i.e.*, $\mathsf{edw}_V(v_i) = \mathsf{util}(v_i)$, in which case $\mathsf{G}_V(v_i) = \sum_{j=1}^{\mathfrak{r}} \mathsf{util}(v_i)_j$. □

3.3.4.3 Uncertainty Incentive

For the subsequent discussion (this includes Section 3.3.4.4), G_V is defined as G_V^{δ} whereat $\frac{4}{\delta+2}$ and $\frac{2}{\delta+2}$ are replaced by $\mathscr{A}$ and $\mathscr{B}$, respectively.

The uncertainty incentive demands, that there is no incentive to configure a VM with too many or too few VRs in case of uncertainty about its upcoming utilization. The uncertainty incentive is provided when enforcing cloud fairness according to Definition 3.6, as Definition 3.3 replaces factors $\mathscr{A}$ and $\mathscr{B}$ from Equation 3.1 by $\frac{4}{\delta+2}$ and $\frac{2}{\delta+2}$ respectively. The proof of the following theorem shows why this replacement works.

Theorem 3.13. *Enforcing cloud fairness according to Definition 3.6 provides the uncertainty incentive.*

Proof. The theorem is proven on a per VM basis. Let $v_i \in V$ be a VM. To measure how well VM v_i is configured, the Unnecessary Greediness Increase (UGI) of v_i is defined as the difference between the greediness of v_i and the lowest greediness possible given v_i's PR utilization, *i.e.*,

$$\mathsf{UGI}(v_i) := \mathsf{G}_V(v_i) - \sum_{i=j}^{\mathfrak{r}} \mathsf{util}(v_i)_j. \tag{3.17}$$

Assume that, when v_i's PR utilization remains z% under v_i's endowment, the UGI is z and, when v_i's PR utilization exceeds v_i's endowment by z%, the UGI is $2 \cdot z$. Thus, configuring v_i with z% too few VRs is twice as "costly" than configuring v_i with z% too many VRs. Thus, users have incentive too configure VMs too large, if they are uncertain about the PR utilization. Consequently, the uncertainty incentive is not provided, as the uncertainty incentive requires, that configuring a VM with z% too many or too few VRs results in the same UGI.

Without loss of generality, assume that there is one PR r and that the DCF is $\mathscr{D}$. Let $x > y > 0$. Let $\hat{v} \in V$ with $\mathsf{edw}_V(\hat{v}) = x$ and $\mathsf{util}(\hat{v}) = x + y$, *i.e.*,

$\hat{v}$ exceeds its endowment of x by y. Then $\mathsf{G}_V(\hat{v}) = x + \mathscr{A} \cdot y$ and $\mathsf{UGI}(\hat{v}) = y \cdot \mathscr{A} - y$ holds. Let $\check{v} \in V$ with $\mathsf{edw}_V(\check{v}) = x$ and $\mathsf{util}(\check{v}) = x - y$, *i.e.*, $\check{v}$ leaves y of its endowment of x unutilized. Then $\mathsf{G}_V(\check{v}) = x - \mathscr{B} \cdot \mathscr{D} \cdot y$ and $\mathsf{UGI}(\check{v}) = y - \mathscr{B} \cdot \mathscr{D} \cdot y$ holds. In order to provide the uncertainty incentive, $\mathscr{A}$ and $\mathscr{B}$ must be determined, such that $\mathsf{UGI}(\hat{v}) = \mathsf{UGI}(\check{v})$. This is the case for

$$\mathscr{A} = 2 - \mathscr{B} \cdot \mathscr{D}, \tag{3.18}$$

where $0 \leq \mathscr{D} \leq 1$, as it represents the DCF. $\mathscr{D}$ has to be estimated by the Cloud Service Provider (CSP) based on the expected cloud utilization, which depends on factors, such as the cloud's overcommit ratios.

When the CSP choses $\mathscr{D}$ incorrectly, the uncertainty incentive is still provided in practice for the following reason. In order to take advantage of an incorrect choice of $\mathscr{D}$, a user needs to know the ratio of the chosen $\mathscr{D}$ and the actual DCF. However, the user has little information to determine the actual DCF. Furthermore, the CSP does not need to announce the chosen $\mathscr{D}$ to the user. Therefore, the user has insufficient information to determine the ratio needed to make strategic VM configurations in case $\mathscr{D}$ is chosen incorrectly.

One solution to Equation 3.18 is $\mathscr{A} = \frac{4}{\mathscr{D}+2}$ and $\mathscr{B} = \frac{2}{\mathscr{D}+2}$ as implemented by $\mathsf{G}_V^{\mathscr{D}}$. Thus, enforcing cloud fairness according to Definition 3.6 provides the uncertainty incentive. □

The next section shows why this particular solution to Equation 3.18 is chosen.

3.3.4.4 Questionnaire Compliance

The proof of Theorem 3.13 established a dependency between $\mathscr{A}$ and $\mathscr{B}$, such that enforcing cloud fairness according to Definition 3.6 provides the uncertainty incentive. It was shown that G_V^{δ} implements a specific solution to this dependency. As the results of Q3 have already been fully exploited to design G_V^{δ} and the results of Q2 cannot be used (cf. Section 3.2.4.2), explicit results from questionnaire's Q1 (cf. Section 3.2.1) lead to this specific solution, as shown subsequently.

A11 and A14 were rejected by the vast majority of participants and are, therefore, not considered further. As A13 was the most frequent choice and

was selected by roughly twice as many participants, as the second frequent choice A12, values for $\mathscr{A}$ and $\mathscr{B}$ are determined, such that the greediness metric qualifies $A13$ as twice as fair as $A12$, whereat fairness of an allocation is quantified by the (maximal) difference of user greediness for this allocation. Because the questionnaire specified real-life scenarios, no endowments were specified. Thus, to apply G_V, all endowments (function edw_V in Equation 3.6) are defined as $(2, 2)$, *i.e.*, the equal share.

For A12, G_V results in $\mathsf{G}_V(c_1) = \mathsf{G}_V(c_2) = \mathscr{A} - 2\mathscr{B}$ and $\mathsf{G}_V(c_3) = 2\mathscr{A}$. For A13, G_V results in $\mathsf{G}_V(c_1) = \mathsf{G}_V(c_2) = 2\mathscr{A} - 2\mathscr{B}$ and $\mathsf{G}_V(c_3) = 0$. Thus, for A12 the greediness range is $\mathscr{A} + 2\mathscr{B}$ and for A13 it is $2\mathscr{A} - 2\mathscr{B}$. Accordingly, $\mathscr{A}$ and $\mathscr{B}$ must be determined, such that the greediness range of A12 is twice the greediness range of A13, *i.e.*,

$$\mathscr{A} + 2\mathscr{B} = 2 \cdot (2\mathscr{A} - 2\mathscr{B}). \tag{3.19}$$

Inserting Equation 3.18 into Equation 3.19 yields

$$\mathscr{A} = \frac{4}{\mathscr{D} + 2} \quad \text{and} \quad \mathscr{B} = \frac{2}{\mathscr{D} + 2}. \tag{3.20}$$

Equation 3.20 and setting $\mathscr{D} = \delta$, finally justifies why G_V^δ uses $\frac{4}{\delta+2}$ and $\frac{2}{\delta+2}$ as factors. $0 \leq \delta \leq 1$ holds, as δ, or rather $\mathscr{D}$, estimates the DCF (cf. Section 3.3.4.3), which is within this range. Therefore, in practice, δ has to increase with the expected cloud utilization levels (the higher these levels, the more likely it is that PRs are ceded and, thus, the DCF estimate has to increase).

δ determines how strongly a deviation of the PR utilization from the endowment is penalized. For $\delta = 0$, G_V^δ penalizes exceeding the endowment most strongly and, in turn, rewards ceding most strongly. The opposite holds for $\delta = 1$. The conformance of G with all relevant results of the questionnaire was verified numerically, when $\mathscr{A} = \frac{4}{\delta+2}$ and $\mathscr{B} = \frac{2}{\delta+2}$ for any $0 \leq \delta \leq 1$. The results show G's perfect compliance with all questionnaire results for any δ. Therefore, also G_V^δ perfectly complies with the questionnaire, when equal endowments are assumed.

3.3.4.5 Comparison to Other Metrics

By designing GM (cf. Definition 3.1) in line with the most frequent arguments made in Q3 of the questionnaire, GM is able to quantify the intuitively perceived greediness of consumers based on the resources they served themselves from a shared resource pool. Therefore, in contrast to the metrics discussed in Section 3.2.3.1, which were frequently suggested for this purpose, GM produces the rankings for all three scenarios of Q3 that were most frequently selected by participants and also identifies the most frequently selected answer to Q1 as most fair.

GM assumes that all consumers have equal entitlements to the resource pool. This assumption was relaxed, when refining GM (cf. Definition 3.3) to be applicable to VMs. This refined GM allows taking very different entitlements to resources into account. Moreover, these entitlements are not captured by simple weights but by vectors, which allows accounting for different entitlements to different resources, which naturally fits the cloud context, where VMs have different entitlements to different PRs.

By adding these entitlements statically to the greediness of a VM and by adding or subtracting a certain number, based on how the actual consumption deviates from these entitlements, the refined GM is able to take into account, how well VMs are configured. This allows providing three incentives to configure VMs' VRs to best match the anticipated subsequent PR utilization. No other metric is able to provide these incentives.

Thus, the refined GM is the first metric that not only captures an intuitive understanding of greediness but also is applicable in a highly technical context, where different entitlements to different resources exist.

3.4 Discussion

Defining fairness is significantly easier, when utility functions of consumers are known. As cloud users utilize resources via VMs, defining cloud fairness would be simpler, if utility functions of VMs are known. Therefore, Section 3.1 broke the first ground in this unexplored research field of charting VM utility functions, by investigating dependencies between VR combinations a VM has access to and the VM's performance. The results obtained were

negative, as it was not possible to find consistent dependencies and even counter-intuitive dependencies were discovered. For example, it was shown that (a) in some cases the performance of a VM decreases with an increasing number of VCPUs and (b) a VM's performance often does not decrease, when it has tight RAM constraints. Although there exist technical explanations for these phenomena, their occurrence prohibits the use of utility functions to define cloud fairness.

These negative results motivated a novel approach to define fairness in Section 3.2. Conceptualizing cloud nodes as "self-serving buffets" and observing that nodes can prioritize VMs' access to their PRs, lead to the premiss that it is fair to constrain consumers in proportion to their greediness. Therefore, instead of fairness, greediness had to be defined. This was done based on a survey among more than 600 participants. This survey (a) posed questions by simple real-life scenarios to not constrain the range of participants or distract them by technical terms, (b) showed that DRF, which is the state-of-the-art of data center fairness, does not comply with an intuitive understanding of fairness, and (c) provided information to formulate the Greediness Metric (GM), which captures an intuitive understanding of greediness without access to consumers' utility functions but solely based on the resources they consume.

The GM was formulated in Section 3.3, which also refined it to define cloud user greediness. In line with the premiss, cloud fairness was defined as prioritizing VMs inversely proportional to the greediness of their user. It was shown that (a) enforcing this definition of cloud fairness provides incentive to users to configure VRs of their VMs correctly, that is, aligned with the PR requirements of their VMs, and (b) for the purpose of prioritizing VMs to enforce fairness, the proposed cloud user greediness definition is superior to any well established metric that quantifies multi-resource consumption.

This superiority will further be confirmed by the use of the simulator developed in Chapter 4. In addition to this simulator, an OpenStack implementation extension developed practically enables the enforcement of this new definition of cloud fairness.

4

Simulation and Implementation Architecture

The simulative comparison of various policies to allocate cloud resources in a fair manner as well as fair cloud resource allocation by adapting node resources allocation to running VMs were unexplored research fields [39, 97].

A cloud simulator that allocates node resources to VMs is developed in Python in Section 4.1. The simulator calculates a snapshot resource allocation that is perfect according to one of ten policies. Policies apply one of five metrics (cf. Section 4.1.2) to quantify how much VMs and users receive and enforce max-min fairness among those quantifications (cf. Section 2.4.3 and 3.3.3.4) either with a per-user or per-VM scope. Section 4.1.3 elaborates on how these allocations are calculated. Fairness of allocations is measured by quantifying bundles with one of these five metrics and feeding the results of these quantifications to Jain's index [43, 97] (cf. Section 4.1.4). Accordingly, there are various allocations and ways to evaluate their fairness. To allow for a

comprehensive discussion of all resulting constellations, the simulator calculates an allocation snapshot, as an allocation that changes over time complicates the discussion, while not allowing for deeper insights. For the same reason, only perfect allocations are considered, as non-perfectness adds varity that complicates the discussion but does broaden it in a reasonable manner. Therefore, the simulator does not evaluate practical but mathematical/theoretical aspects and compares different metrics to prioritize VMs.

In order to evaluate practical aspects an OpenStack [73] Fairness Service (FS) is developed in Section 4.2. The FS adapts VMs' PPs to enforce fairness on a per user basis by allocating CPU, RAM, disk I/O, and network access (cf. Section 4.2.5). OpenStack is an open-source cloud computing software that is the de facto standard for private clouds and also serves as foundation for public clouds. OpenStack consists of several different components with one of the most important components being Nova (compute) [69]. Nova runs on every node that hosts VMs and has direct control over these. Therefore, the FS is implemented by an additional nova service called nova-fairness (cf. Section 4.2.1).

4.1 Simulator

The cloud simulator (a) calculates a static/snapshot resource allocation that is max-min fair on a per-user (cf. Theorem 3.9) or per-VM (cf. Corollary 3.10) basis and (b) quantifies the fairness of this allocation [82]. The simulator models different heterogenous resources and deploys different metrics (cf. Section 4.1.2) to make bundles comparable. Making bundles comparable is necessary to apply max-min fairness (cf. Section 2.4.3) and to quantify the fairness of allocations. A policy is defined by (a) a per-user (cf. Definition 3.7) or per-VM (cf. Definition 3.8) scope and (b) the metric that is applied to make bundles comparable. The main entities modeled by the simulator are users, VMs, and nodes. Setups are constellations of these entities. Section 5.2 evaluates different policies by comparing the allocations they result in for fixed setups.

Formally, a *setup* is defined analogous the representation of a cloud as introduced in Section 3.3.2.1. In particular, a setup is a set of users $U =$

$\{u_1, u_2, \ldots, u_u\}$, a set of nodes $N = \{n_1, n_2, \ldots, n_n\}$, a set of VMs $V = \{v_1, v_2, \ldots, v_v\}$, and a set of resources $R = \{r_1, r_2, \ldots, r_r\}$. All functions defined in Section 3.3.2.1 and illustrated in Figure 3.14 also exist for setups. Only function $\mathsf{util}\colon V \to \mathbb{R}^r_{\geq 0}$ is replaced by function $\mathsf{req}\colon V \to \mathbb{R}^r_{\geq 0}$ and denotes the resources a VM requests (but not necessarily receives). Thus, $\mathsf{req}(v)$ is also referred to as v's requirement vector and resources have to be allocated to v in the ratio of this vector. That is, VMs have Leontief utility functions (cf. Section 2.3.3).

While Leontief utility functions do not apply in practice (cf. Section 3.1), it is the standard assumption to model data center resource allocation theoretically [12, 13, 24, 30, 104, 105] and adopted by the simulator to allow for a comprehensive and concise comparison of different policies. Nonetheless, the simulator's abstraction level is more detailed than the abstraction level usually applied in literature as the simulator allows users to own different VMs, each with a different Leontief utility function, while in literature only one Leontief utility function is assumed per user. Therefore, a setup is defined by sets U, N, V, and R and functions own, host, nbr, phy, quota, virt, and req.

A VM's owner is denoted by superscript and a VM's host is denoted by subscript. For example, VM $v^a_x \in V$ is owned by $u_a \in U$ and hosted on $n_x \in N$. If a node hosts more than one VM of the same user, the VMs are distinguished by dots, *e.g.*, $\dot{v}^a_x$ and $\ddot{v}^a_x$.

An allocation for a setup is defined by function $\mathsf{alloc}\colon V \to \mathbb{R}^r_{\geq 0}$ that maps each VM v to the bundle v receives. Accordingly, equation

$$\forall n \in N\colon \sum_{v \in \mathsf{host}(n)} \mathsf{alloc}(v) \leq \mathsf{phy}(n)$$

holds, as the amount of resources allocated to VMs that share a node is limited by the node's resources. To ease the discussion, it is assumed that nodes are not overcommitted, *i.e.*,

$$\forall n \in N\colon \sum_{v \in \mathsf{host}(n)} \mathsf{virt}(v) \leq \mathsf{phy}(n) \tag{4.1}$$

As Equation 3.5 is applied to calculate endowments, Equation 4.1 implies

$$\forall v \in V\colon \mathsf{virt}(v) = \mathsf{edw}_V(v). \tag{4.2}$$

A VMs resource requests are allowed to exceed the VM's VRs, as otherwise overload on nodes would be impossible.

4.1.1 Assumptions and Practice

Section 2.3.3 introduced Leontief utility functions and discussed their suitability to model job scheduling. The simulator uses Leontief utility functions to model node resource allocation and calculates a snapshot resource allocation, *i.e.*, VM demands are static and for these demands a static allocation is calculated.

Leontief utility functions model dependencies among time-shared PRs well. For example, when the bandwidth of a server is reduced, the server's disk I/O utilization decreases equivalently, as requests reach the server slower and accordingly stored data is fetched at a lower rate. In contrast, the dependency between time-shared PRs and the space shared PR RAM is often different. For example, assume a program has a fixed number of variables that are stored in RAM. Allocating more CPU time will execute the program faster, while the number of variables and, thus, required RAM remains constant. Substitutabilities further complicate dependencies among PRs: For example, (a) paging reduces RAM requirements by use of CPU time or (b) compression saves storage or bandwidth by use of CPU time [23]. Consequently, Leontief utility functions do not cover the whole range of PR dependencies.

4.1.2 Metrics

The following metrics are evaluated in the simulations C, D, G^0, $G^{0.5}$, G^1. C, D, and G abbreviate cost, DRF, and greediness, respectively. Using these abbreviations instead of spelling out those names, allows for a concise and compact notation in formulas, figures, and tables. The metrics are used by the simulator to generate allocations and also to quantify the fairness of allocations. All metrics normalize the value of resources according to Section 3.3.1.3 and

serve to map bundles of VMs or users to a number in $\mathbb{R}$. Let $v \in V$, $u \in U$, and $\mathsf{Met} := \{\mathsf{C}, \mathsf{D}, \mathsf{G}^0, \mathsf{G}^{0.5}, \mathsf{G}^1\}$ be the set of metrics. Metric $\mathsf{M} \in \mathsf{Met}$ defines the M-value of v and u as follows.

4.1.2.1 Cost-metric (C-metric)

The cost-metric, abbreviated by C-metric, applies the L_1 norm (cf. Section 3.2.3.1.1) to define the C-value of users and VMs. Therefore, the C-value of v is defined by applying the L_1 metric to v's bundle. The C-value of u is defined by applying the L_1 norm to the sum of bundles of u's VMs. Equivalently, the C-value of u can be defined by the sum of C-values of u's VMs. The C-metric is the only metric considered subsequently, for which this equivalence holds. The L_1 norm is also used to define asset fairness [30], wherefore asset-fairness also would have been a viable name for this metric.

4.1.2.2 DRF-metric (D-metric)

The DRF-metric, abbreviated by D-metric, applies the L_∞ norm (cf. Section 2.4.3 and 3.2.3.1.4) to define the D-value of users and VMs. In particular, the D-value of v is the dominant share of v's bundle. The D-value of u is the dominant share of the sum of bundles of u's VMs. Therefore, the D-value of u may be smaller than the sum of D-values of u's VMs. The D-metric is equivalent to the DoRe metric in Section 3.2.3.1.4.

4.1.2.3 Greediness-metric (G^δ-metric)

The greediness-metric, abbreviated by G^δ-metric, defines the G^δ-value of v by Definition 3.3 and the G^δ-value of u by Definition 3.5 for $\delta \in \{0, 0.5, 1\}$. Therefore, the G^δ-value of u may be negative. When the superscript is omitted from G, the statement/equation holds independent of parameter δ. The G^δ-value of v or u is also referred to as δ-greediness of v or u.

4.1.3 Policies and Allocations

Policies were introduced by Definition 3.7 and 3.8 and Section 3.3.3.4 discussed how they determine max-min fair allocations, when VM demands are stable. This section revisits notations and findings to show how the simulator applies policies.

Every policy is determined by a metric $\mathsf{M} \in \mathsf{Met}$ and a *scope*, which is local or global (cf. Definition 3.7 and 3.8). Accordingly, a policy is denoted as M_s-policy, where $s = l$, if the policy's scope is local, and the $s = g$, if the policy's scope is global. The M_s-policy determines the M_s-allocation, such that (a) VMs receive resources in the ratio of their demand vector, (b) no VM receives more resources than requested, and (c) max-min fairness is enforced among the M-values of users or VMs (cf. Theorem 3.9 and Corollary 3.10). When the policy's scope is local, max-min fairness is enforced on each node among the VMs' M-values, *i.e.*, the owner of VMs is ignored and allocations on different nodes are independent. When the policy's scope is global, max-min fairness is enforced among the users' M-values, *i.e.*, the allocations on different nodes are interdependent. Therefore, the M_g-allocation is M-max-min fair among users and the M_l-allocation is M-max-min fair among VMs. If the subscript is omitted from a policy or allocation, the scope is irrelevant, *i.e.*, the M_l- and M_g-policy result in the same allocation. The G_g^δ-policy implements Definition 3.6.

4.1.3.1 Allocation Calculation

The M_g-policy determines M_g-allocation by the algorithm described in Listing 4.1. Line 1 and 2 define that the temporary allocation alloc^t and the the main allocation alloc^m allocate every VM an empty bundle. The M-value of a user or VM $a \in U \cup V$ according to allocation alloc^m and alloc^t are denoted by $\mathsf{M}^m(a)$ and $\mathsf{M}^t(a)$, respectively. These allocations are adapted by the while loop in Line 3, which runs until the difference between these two allocations does not change by more than 0.001% for 5 iterations. Line 4 merges alloc^m and alloc^t, where the weight of alloc^m in this merge increases every iteration (cf. Line 5). Line 6 updates $\mathsf{M}^m(u)$ for all users and Line 7 "resets" allocation alloc^t. Line 8 iterates over all nodes. The while loop in Line 9 changes allocation alloc^t and iterates as long as alloc^t has not allocated all resources of a node and VMs request more resources. In order to align the M-values, Line 10 selects a VM that requests more resources and most "deserves" them. For $v \in V$, this deserving is determined by

$$\mathsf{M}^*(v) := \mathsf{M}^t(v) - \mathsf{M}^m(v) + \mathsf{M}^m(\mathsf{own}(v)). \tag{4.3}$$

```
1  ∀v ∈ V: alloc^t(v) := {0}^r
2  ∀v ∈ V: alloc^m(v) := {0}^r
3  while(∃v ∈ V, ∃1 ≤ i ≤ r: |alloc^t(v)_i − alloc(v)_i| > 10^−5 in last five
       iterations)
4      alloc^m := (1 − x) · alloc^m + x · alloc^t
5      x := x · 0.995
6      u ∈ U: update M^m(u)
7      ∀v ∈ V: alloc^t(v) := {0}^r
8      ∀n ∈ N do:
9          while( ∀r_i ∈ R: ∑_{v∈host(n)} alloc^t(v)_i < phy(n)_i and
                   v ∈ host(n): alloc^t(v) < req(v)):
10             v := argmin_{v∈{v∈host(n)|alloc^t(v)<req(v)}} M*(v)
11             alloc^t(v) := alloc^t(v) + ε · req(v)/∑_{i=1}^r req(v)_i
12             ∀v ∈ host(n): update M^t(v)
```

Listing 4.1: Determining the M_s-allocation

$M^t(v)$ has to be contained in $M^*(v)$, as $alloc^t$ is the allocation being modified and a VM that deserves more resources most, *i.e.*, currently receives the lowest M-value, has to be selected. $M^m(v)$ contributed in Line 6 to $M^m(own(v))$ and this value influences the resource allocation on all nodes that host VMs of $own(v)$. To stabilize v's contribution to $M^m(own(v))$ over different iterations of the loop in Line 3, the loop in Line 9 must ensure that v is allocated roughly the same resources as in the last iteration of Line 3. Subtracting $M^m(v)$ in Equation 4.3 ensures exactly that, as v is likelier to be selected, when $M^*(v)$ is lower. As $alloc^m$ shall align the M^m-value of all users, VMs of users' that already have a high M^m-value must be less likely to be selected. Therefore, $M^m(own(v))$'s M-value is added in Equation 4.3.

After the VM v that desrves more resources most is determined, Line 11 increases $alloc^t(v)$ by a fraction of $req(v)$, which ensures, that v receives resources in the same ration as $req(v)$. The division by $\sum_{i=1}^{r} req(v)_i$ ensures that the amount by which a VM's bundle is increased, is normalized among VMs. Lastly, Line 12 updates the M^t-value of all VMs hosted by the node. Updating all VMs is important, as changing the bundle of one VM can change the M^t-value of other VMs, if $M = G$.

4.1.3.1.1 Implementation Details

An implementation detail that is not shown in the pseudo code, is how the approximation of an allocation depicted in Line 11 by ε is implemented. In particular, the loop in Line 9 actually runs until (a) some resource r_s is "highly" saturated, *i.e.*,

$$\exists r_s \in R\colon \sum_{v \in \mathsf{host}(n)} \mathsf{alloc}^t(v)_s > \mathsf{phy}(n)_s - \varepsilon', \tag{4.4}$$

(b) no resource is over-saturated, *i.e.*,

$$\forall r_a \in R\colon \sum_{v \in \mathsf{host}(n)} \mathsf{alloc}^t(v)_a \leq \mathsf{phy}(n)_a, \tag{4.5}$$

and (c) the difference among the M^*-values of all VMs requesting more resources is at most ε', *i.e.*,

$$\max_{v_a, v_c \in \{v \in \mathsf{host}(n) \mid \mathsf{alloc}^t(v) < \mathsf{req}(v)\}} |\mathsf{M}^*(v_a) - \mathsf{M}^*(v_c)| < \varepsilon', \tag{4.6}$$

where $\varepsilon' = 10^{-9}$. This value of ε' ensures that the determined allocation is equal to a perfect allocation, *i.e.*, an allocation where equality holds in Equation 4.4 and 4.6 for $\varepsilon' = 0$, in the first four decimal digits; an accuracy that is more than sufficient for all discussion based on the simulator's results (cf. Section 5.2). To determine such allocations, resources are allocated to the VM v with smallest $\mathsf{M}^*(v)$ (cf. Line 10 and 11), if no resource is over-saturated, and resources are removed from the VM v with largest $\mathsf{M}^*(v)$, if a resource is over-saturated. This is implemented by initializing ε (cf. Line 11) with 0.05 and multiplying it at beginning of the loop in Line 9 with -0.9, if (a) a resource is over-saturated and $\varepsilon > 0$ or (b) no resource is saturated and $\varepsilon < 0$. Therefore, $|\varepsilon|$ decreases and alloc^t oscillate around and converges towards an optimal allocation. Another implementation detail not shown here is the handling of VMs that request zero of some resource: VMs that request zero of r_s (cf. Equation 4.4) have to be allocated more resources to make the allocation efficient, while Equation 4.6 has to be verified only for those VMs that still can be allocated resources.

4.1.3.1.2 Local Policies

As for local policies allocations on different nodes are independent and since the M-values of users are not considered, local allocations are generated by running only Lines 8 to 12 of Listing 4.1 where

$$\mathsf{M}^*(v) := \mathsf{M}^t(v). \tag{4.7}$$

4.1.4 Fairness Score and Fairness Quantification

While the quantification of single-resource allocations is well researched (cf. [43, 51, 90] and Section 2.6.10.1), there is no best practice to quantify the fairness of multi-resource allocations. In particular, single-resource fairness returns a real number that quantifies the fairness of an allocation, which is specified by vector $v \in \mathbb{R}^{\mathfrak{c}}_{\geq 0}$, where v_i is what user u_i receives. Therefore, to apply a single-resource fairness measure to a multi-resource allocation, the bundle of each user has to be mapped to a scalar, such that the resulting vector of scalars can be input to the single-resource fairness measure.

[44] suggests the dominant share or the number of jobs a user is able to schedule, to map a multi-resource allocation to a scalar. As a single-resource fairness measure, [44] suggests a function of [51] that is parameterized by $\beta, \lambda \in \mathbb{R}$.

This thesis quantifies the fairness of multi-resource allocations by

$$\mathbb{R}^{\mathfrak{r} \times \mathfrak{u}} \xrightarrow{\mathsf{M} \in \mathsf{Met}} \mathbb{R}^{\mathfrak{u}} \xrightarrow{\text{Jain's Index}} [0, 1]. \tag{4.8}$$

In particular, the metrics discussed in Section 4.1.2, *i.e.*, the metrics in Met, are used to map a multi-resource allocation to a scalar. As single-resource fairness measure, this thesis uses Jain's Index [43], as it is the traditional fairness measure and a special case of the function suggested in [44, 51]. As $|\mathsf{Met}| = 5$ this results in five fairness measures (one for each metric in Met) applied to the allocations.

Jain's Index takes a vector of non-negative numbers as input, while the subtraction of the quota credit may lead to negative G-values. To avoid negative values as input for Jain's index, the largest quota credit among users is de-

termined and added to the G-value of every user before being input to Jain's index.

Using the D-metric in the left mapping of Equation 4.8, quantifies bundles according to the dominant share as suggested in [44]. [44] alternatively suggests to quantify bundles, by the number of jobs users are able to schedule. This quantification is not used here, as (a) two users may be able to schedule different numbers of jobs with the same amount of resources, and therefore, the same bundle is quantified differently for them, and (b) the simulator does not model jobs.

For metric $\mathsf{M} \in \mathsf{Met}$, the M-score of allocation A is defined as the result of applying Equation 4.8 to A, where M is used in the left mapping of the equation.

4.2 OpenStack Extension

OpenStack [42] is the de facto standard for private cloud software stacks and also serves as a foundation for public clouds. OpenStack is open source and has a large, supportive community that organizes regular meetings and summits all over the world. Out of the box, OpenStack does not control cloud resources by adapting PR allocations, but the idea was well received, when discussed at the 13th Swiss OpenStack User Group Meetup on July 7th, 2016, in Winterthur.

This section describes the extension of OpenStack to leverage PR allocation in order to enforce fairness among cloud users [64, 79, 80]. Fairness is enforced as discussed in Section 3.3.3.

4.2.1 High-level Design and Steps

Two essential node types in the OpenStack architecture are the controller node and the compute nodes. The *controller node* coordinates the cloud, *e.g.*, by scheduling VMs, and stores essential information to be accessed by other nodes or administrators. *Compute nodes* perform the actual processing of workloads, *i.e.*, hosting VMs. For this purpose, compute nodes run the OpenStack service `nova-compute` that is part of the OpenStack compute project nova. OpenStack nova knows three types of managing components: Ser-

```
1  while NRI of some node in N^f is missing:
2      send own NRI to nodes of which NRI is missing
3  use NRIs to calculate CRS and normalization vector
4  every μ seconds:
5      collect RUI of all VMs hosted by n_i
6      apply G_V^δ to collected RUI in order to calculate greediness of all VMs
           hosted by n_i
7      send this greediness set to all n ∈ N^f − {n_i}
8      wait to receive greediness set from all n ∈ N^f − {n_i}
9      apply G_U^δ to calculate the greediness of all u ∈ U
10     for every VM v hosted by n_i:
11         set priorities of v according to G_V^δ(v) and G_U^δ(host(v))
```

Listing 4.2: Steps of the FS running on node n_i

vices, managers and drivers [72]. Services are generic containers that group compute node functionalities. Each service has its own managers to control a specific aspect of the node and execute tasks. Managers encapsulate functionalities of the service by providing methods and periodic tasks to deploy the functionality.

A compute node can monitor the RUI of the VMs it hosts and adapt their priorities. Thus, PR allocations are adapted by an additional nova service called `nova-fairness`. This Fairness Service (FS) enforces fairness in the cloud as discussed in Section 3.3.3 (accordingly, it collects all information discussed in Section 3.3.2.1). The message exchange between FS instances on different nodes is decentralized.

Let $N^f \subseteq N$ be the set of compute nodes that run the FS and $n_i \in N^f$. The pseudocode in Listing 4.2 describes how the FS running on node n_i calculates the greediness of users and adapts the priorities of hosted VMs.

Lines 1 to 3 ensure that the CRS, which is essential to calculate the greediness, is available before the FS conducts any further steps. In order to allow adding nova-fairness nodes subsequently, the FS responds with its NRI upon receiving the NRI of a new node (this is not reflected in the pseudo code). μ in Line 4 defines the interval with which the greediness of users and PPs of VMs are updated and is referred to as the *update interval*. Every μ seconds the

FS calculates the greediness of VMs hosted by n_i (Line 5 and 6). This *greediness set* is announced to all nodes (Line 7). When greediness sets have been received from all other nodes (Line 8), the node calculates the greediness of users (Line 9) from this information. Lastly, priorities of VMs on n_i are set according to the calculated greediness (Line 10 and 11).

4.2.2 Resource Measurement

The FS takes the following six PRs into account: (a) CPU time in seconds, (b) memory used in kilobyte, number of bytes (c) read from disk and (d) written to disk, and number of bytes (e) received and (f) transmitted through the network interface. Disk space and RAM bandwidth are not considered by this prototypical implementation, as disk usage is accounted for by (c) and (d) and RAM usage is accounted for by (b). Resources such as GPUs and software licenses are not taken into account as they are relevant only in specific scenarios. Therefore, the FS takes all PRs that are relevant for generic computing tasks and most relevant in the context of clouds into account (cf. Section 2.2.1).

4.2.2.1 CPU Time Normalization

CPU time, *i.e.*, the amount of time used for a specific CPU task to complete [99], measures CPU usage. CPU time provided by different nodes is not directly comparable. The reason is that cloud nodes are rarely homogeneous [36] and, therefore, are equipped with different CPUs. Accordingly, one second of CPU time on a powerful node is more valuable than one second of CPU time on a less powerful node. To compare CPU time across nodes, the FS normalizes CPU time by the nodes' BogoMIPS [110]. BogoMIPS is a metric provided by the Linux operating system to capture the performance of different CPUs. However, BogoMIPS do not define a scientifically reliable measure to compare CPUs, wherefore other normalization references, such as the SPEC value [96], are considered for future improvements.

4.2.2.2 Resource Utilization Information

The FS requests a list of VMs that are running on its node by an Remote Procedure Call (RPC) to the `nova-conductor` service. This list contains

only VMs that have been successfully scheduled by OpenStack on the node. Therefore, the list indicates for which VMs RUI have to be collected (cf. Line 5 of Listing 4.2). The RUI is collected by deploying an OpenStack driver to access the `libvirt` virtualization API [5]. This API allows monitoring the detailed VM RUI in a unified manner and supports most of the hypervisors known [87]. Accordingly, the libvirt API provides access to the RUI for each VM and ensures the FS's compatibility with numerous hypervisors. For time-shared PRs (CPU time, disk I/O, network access), the libvirt API provides the accumulated PR utilization since boot time. Therefore, the FS calculates the RUI for the current update interval by subtracting the accumulated utilization at the beginning of the interval from the accumulated utilization at the end of the interval. For space-shared PRs (RAM) the libvirt API provides the current utilization, which the FS uses to represent RAM utilization in the RUI vector.

VMs' PR utilization may follow well defined but bursty patterns. For example, databases usually write in periodic large bursts to disk. However, users cannot specify this burstiness, as they only configure VRs of their VMs. Nonetheless, the FS does not penalize bursty PR utilization, when enforcing fairness. The reason is that the FS measures the PR utilization of time-shared PRs accumulated over period μ. Therefore, it does not matter, if within this period PRs are utilized smoothly or in bursts. Only when the inter-burst time is greater than μ, bursts are penalized.

4.2.2.3 Node Resource Information

The NRI of a node specifies its PR capacities. Therefore, the NRI per node is stable, which allows to capture NRI without a temporal component (Line 1-3 of Listing 4.2). The different NRI data points considered by the FS are (a) number of CPU cores, normalized by the node's BogoMIPS, (b) combined disk read speeds of all disks in bytes/s, (c) network throughput in Byte/s, and (d) total amount of installed memory in kBytes.

4.2.3 Metric Customization

The FS enforces fairness according to Definition 3.6. However, the FS allows to overwrite the G_U^δ and G_V^δ used by this definition. The definitions to be

used instead are specified in the nova configuration by the class path of these definitions. This class path is checked by the FS for correctness, *i.e.*, whether a correct class with required definitions exists under that path. Therefore, the FS allows to enforce fairness according Definitions 3.7 and 3.8, which includes fairness definitions such as asset fairness or DRF.

Flavors in OpenStack do not contain a virtual counterpart for every PR. For example, OpenStack flavors contain VCPU and VRAM but not virtual disk I/O or virtual network access. However, the definitions for function edw_V in Section 3.3.2.3 assume that every PR has a virtual counterpart. Therefore, in case a PR has none, the FS divides its node's supply of that PR by the number of hosted VMs. In turn, this result determines the amount of the virtual counterpart of this PR every VM on that node owns. For example, when a node with a bandwidth of 10 Gbit/s hosts four VMs, each VM's virtual network access is set to 2.5 Gbit/s.

4.2.4 Message Exchange

The FS's information exchange between compute nodes (cf. Lines 2, 7, and 8 of Listing 4.2) is implemented in a decentralized and asynchronous manner. Section 5.4 compares two alternative types of message flow topologies, discusses and compares their details and complexity, and provides according illustrations.

4.2.4.1 Decentralization

OpenStack enables message exchanges between nodes by message queues. OpenStack's default message queuing protocol is `RabbitMQ` [20], which is an implementation of the `AMQP` specification. AMQP is centralized, wherefore using RabbitMQ implies that all inter-node communication traverses the RabbitMQ broker on the controller node that relays messages. To allow for a decentralized information exchange between compute nodes, the `ZeroMQ` [70] message queuing system is deployed. ZeroMQ can be set up with minimal effort and allows compute nodes to exchange information directly by running a decentralized ZeroMQ message broker on each node.

The only mechanism of ZeroMQ that works centralized is the discovery of nodes to communicate with. For this purpose, a `redis` data structure

store [88] hosting identification information for all ZeroMQ brokers has to be installed on the controller node. Because redis is an in-memory data store, it is highly performant and sufficiently scalable.

4.2.4.2 Isochronous Message Exchange

To achieve independence of compute nodes, FS instances on different nodes operate isochronously, *i.e.*, nodes adhere to the same update interval μ, but send messages within this interval at different times. Moreover, messages between nodes are sent by RPC casts, as these, in contrast to calls, do not invoke a response. Accordingly, messages that contain NRI and greediness sets from other nodes reach a node at different times. To ensure that messages are only sent to nodes, which are currently reachable and running the FS, an according list is requested first from the `nova-conductor` service. The nova-conductor queries a `SQL` database located on the controller, which contains all status information about services currently running.

The NRI is static, wherefore a compute node only needs to receive this information once from every other node. Therefore, the FS waits to conduct any other operations, until the NRI of all other nodes have been received (cf. Lines 1-3 in Listing 4.2). In contrast to the NRI, VM greediness constantly changes and, therefore, each greediness set that a compute node sends is associated with a certain period of time. Thus, this information needs to be synchronized, when it reaches a compute node. To do so, every compute node n_x maintains a FIFO queue for every other compute node n_z that stores greediness sets received from n_z. When all of these queues contain at least one element, n_x dequeues the first element from every queue to recalculate the user greediness and update VM priorities.

4.2.5 Calculating and Applying Priorities

The FS allocates PRs to VMs by PPs (cf. Section 2.2.1.1, Lines 10 and 11 of Listing 4.2, and [35]). All PRs except network access are controlled by libvirt, wherefore the FS supports most of the known hypervisors [87]. The FS calculates PPs of a VM v_i based on the number $\mathsf{G}_p^\delta(v_i) := \mathsf{G}_V^\delta(v_i) + \mathsf{G}_U^\delta(\mathsf{own}(v_i))$. Number $\mathsf{G}_U^\delta(\mathsf{own}(v_i))$ already includes $\mathsf{G}_V^\delta(v_i)$ (cf. Equation 3.9). However, by adding $\mathsf{G}_V^\delta(v_i)$ again, the individual greediness of the VM

is emphasized, when setting the VM's PPs (otherwise all VMs of a user would have the same PPs). Generally, the ranges of PPs differ among PRs:

CPU time is controlled by setting PPs, alias CPU shares, in the range [1,100].

RAM is space-shared and not time-shared. Thus, it is not allocated by PPs. In turn, the FS uses soft limits, which are a minimum guarantee. The FS assigns soft-limits from 10 MiB to the VMs' maximum amount of RAM.

Disk I/O is controlled by setting PPs, alias disk weights, in the range [100,1000]. This is the maximal range allowed by libvirt.

Network access is the only PR that is not controlled by libvirt, as libvirt only allows setting hard limits for network access, *i.e.*, a maximum bandwidth that cannot be exceeded even if no other VM produces traffic. To avoid the corresponding potential waste of bandwidth, the FS currently deploys the more sophisticated Hierarchical Token Bucket (HTB) queuing discipline of `tc` by calling tc's corresponding Command Line Interfaces (CLI). This allows for setting PPs in the range [1, 98]. Section 5.3.2 integrates the Hierarchical Fair Service Curve (HFSC) queuing discipline into libvirt, such that the FS can set PPs for network access via libvirt.

Function

$$\mathsf{p}(g) \mapsto \begin{cases} \mathrm{out}_{\max} & \text{if } g \geq \mathrm{in}_{\max}, \\ \mathrm{out}_{\min} & \text{if } g \leq \mathrm{in}_{\min}, \\ \frac{(g-\mathrm{in}_{\min})\cdot(\mathrm{out}_{\max}-\mathrm{out}_{\min})}{\mathrm{in}_{\max}-\mathrm{in}_{\min}} + \mathrm{out}_{\min} & \text{else}, \end{cases} \tag{4.9}$$

translates $g := \mathsf{G}_p^{\delta}(v_i)$ to PPs. In particular, greediness g is expected to be in interval $[\mathrm{in}_{\min}, \mathrm{in}_{\max}]$ and $[\mathrm{out}_{\min}, \mathrm{out}_{\max}]$ is the interval of the generated output values. If $g \notin [\mathrm{in}_{\min}, \mathrm{in}_{\max}]$, the according maximum/minimum value is returned. Otherwise, it is calculated where g lies relatively in interval $[\mathrm{in}_{\min}, \mathrm{in}_{\max}]$. The value in interval $[\mathrm{out}_{\min}, \mathrm{out}_{\max}]$ that lies relatively at the same position is returned. For example, when g lies right in the middle of

in_{min} and in_{max}, the value that lies right in the middle of out_{min} and out_{max} is returned.

Values in_{min} and in_{max} have to be defined depending on the metric that is used by the FS. By default the G^{δ}-metric is used for which $g < 0$ is possible and, thus, a $in_{min} < 0$ should be chosen. However, if the G^{δ}-metric is overwritten (cf. Section 4.2.3) by, *e.g.*, the C- or D-metric, $g \geq 0$, wherefore $in_{min} = 0$ is a good choice. Because all metrics normalize resources, reasonable values of in_{min} and in_{max} are easily determined. For the experimental evaluation in Section 5.3, $in_{min} = -1$ and $in_{max} = 1$ was chosen, as Section 5.2 will show that the greediness values usually fall within that range (although it is possible to construct extreme cases where input values $\notin [in_{min}, in_{max}]$ occur).

The priority values that can be assigned to a VM to access a PR differ depending on the PR. Therefore, interval $[out_{min}, out_{max}]$ has to be defined depending on the PR to be controlled. As discussed above, the FS uses intervals $[1, 100]$, $[100, 1000]$, and $[1, 98]$ for CPU time, disk I/O, and network access, respectively. In particular, these intervals are dictated by priority ranges that the libvirt API allows to assign to these PRs.

When the FS updates the PPs of VMs, an exponential decay is used, to avoid fast changing of PPs. In particular

$$\text{updated PP} := (1 - \mathscr{E}) \cdot \text{current PP} + \mathscr{E} \cdot p(h), \tag{4.10}$$

for $0 \leq \mathscr{E} \leq 1$, where for the experimental evaluation in Section 5.3 $\mathscr{E} = 0.2$ is chosen. This value was determined to provide a good tradeoff between smoothness and promptness of the adaption of priorities.

4.3 Discussion

This chapter developed a cloud simulator that calculates max-min-fair multi-resource allocations while using different bundle measures (cf. Section 2.4.3). The simulator models every VM by a Leontief utility function, which is generally too simplistic to capture resource requirements in clouds. The decision to use Leontief utility functions was made, because they are the best

option currently available to model data center multi-resource allocation. In particular, Leontief utility functions are most prominently used in literature (cf. Section 2.6) and capture dependencies between resources in a mathematically simple manner, as dependencies between resources are linear. An advantage of the simulator over many multi-resource allocation models in literature is that the simulator models (a) multiple VMs per user and (b) that VMs are located on different nodes. In contrast, the work introducing DRF [30], which defines today's state-of-the-art of fair data center multi-resource allocations, assumes a sinlge Leontief utility function per user (not modeling different VMs per user) and their competition for resources in a monolithic resource pool (not modeling that these resources are partitioned to nodes).

Also OpenStack was extended by a fairness service that enforces cloud fairness as proposed in this thesis by conducting three steps: (a) monitoring the PR utilization of VMs, (b) periodically exchanging this information across nodes to calculate the greediness of cloud users, and (c) periodically assigning priorities to VMs inversely proportional to the greediness of their owner. PRs that are monitored in Step a and for which priorities are assigned in Step c are CPU, disk I/O, network access, and RAM. The libvirt API is used for both of these steps, making the fairness service compatible to a multitude of hypervisors. For the message exchange in Step b OpenStack's standard message queues are used.

Thus, Chapter 5 does not only evaluate the solution developed, but (a) improves the messaging scheme used in Step b, (b) evaluates the overhead caused by the fairness service, and (c) shows that it effectively enforces fairness among cloud users. Furthermore, the simulator is applied to compare different bundle measures in terms of their suitability to extend max-min-fairness to cloud multi-resource allocations.

5 Evaluation

This thesis' proposal to establish cloud fairness is evaluated as follows. A theoretical evaluation compares the G^{δ}-, D-, and C-policies, with respect to their suitability to define cloud fairness. This theoretical evaluation is constituted by an analytical investigation in Section 5.1 and a simulative investigation in Section 5.2. The analytical investigation begins by investigating the effect of the G^{δ}-policy's δ-parameter and points out frequent cases in which it has no effect on the allocation. Subsequently, it is shown that under simplifying assumptions the G^{δ}-policy is (a) strategy-prove, (b) envy-free, (c) Pareto-efficient, and (d) provides sharing incentives. Under the assumptions made also the D- and C-policies exhibit these properties and, thus, the analytical investigation results in a tie between the policies in terms of their suitability for clouds. This tie is broken by the simulative investigation that follows. The first simulations show effects of different values for the G^{δ}-policy's δ-parameter and, thereby, complement the analytical results. Next, it is shown how the G^{δ}-policy provides the configuration incentive. This incentive is neither provided by the D- nor the C-policy. Subsequent simula-

tions show how the three policies compare to one another in case of multiple nodes. The section closes by summarizing all theoretical results. As these results show the G^{δ}-policy's, *i.e.*, Definition 3.6's, superiority, the G^{δ}-policy is integrated into the Fairness Service (FS).

This FS is investigated by the practical evaluation, which shows the FS's functionality and, thereby, proves that the approach put forward in this thesis is technically feasible. Just as the theoretical evaluation, the practical evaluation consists of two parts: an experimental investigation in Section 5.3 and a structural investigation in Section 5.4. The experimental investigation shows how the FS changes the priorities of running Virtual Machines (VM) to enforce fairness. This makes the FS more potent to control cloud resources than VM scheduling. The structural investigation shows how the message volume produced by the FS can be reduced to an amount that for an individual node only increases with the number of users. This proves the FS's high scalability.

5.1 Analytical Investigations

Section 5.1.1 investigates the effect of the G^{δ}-policy's δ-parameter to point out frequent cases in which δ has no effect on the allocation. Section 5.1.2 shows that under simplifying assumptions the G^{δ}-policy is (a) strategy-prove, (b) envy-free, (c) Pareto-efficient, and (d) provides sharing incentives. Section 5.1.3 summarizes the results of the analytical investigation.

5.1.1 Effect of δ

When fairness is investigated, it often makes sense to assume that all users are equal, *i.e.*, have the same quota. Furthermore, it often makes sense to assume that all users deploy an equal amount of their quota to instantiate VMs, as otherwise the configuration and uncertainty incentives would demand to allocate users different amounts of resources, which complicates the discussion of fairness. To simplify the discussion, simulation setups are designed such that Equation 4.2 holds, which is the case, when nodes are not overcommitted (cf. Definition 3.2). Notably, when these assumptions hold, the G^{δ}_{g}-allocation is independent of parameter δ, as proven next.

Theorem 5.1. *When (a) all users have the same quota,* i.e., $\forall u_a \in U\colon \mathsf{quota}(u_a) = q$ *for some* $q \in \mathbb{R}^{\mathfrak{r}}$*, (b) all users deploy the same percentage of their quota,* i.e., $\forall u_a \in U\colon \sum_{v^a \in \mathsf{own}(u_a)} \sum_{j=0}^{\mathfrak{r}} \mathsf{edw}_V(v^a)_j = s$ *for some* $s \in \mathbb{R}$*, and (c) Equation 4.2 holds, then the* G_g^δ*-allocation is independent of* δ*.*

Proof. From Definition 3.5 and 3.3 follows that for user $u_a \in U$

$$\begin{aligned}\mathsf{G}_U^\delta(u_a) = &\sum_{v^a \in \mathsf{own}(u_a)} \sum_{j=1}^{\mathfrak{r}} \mathsf{edw}_V(v^a)_j + \\ &\sum_{v^a \in \mathsf{own}(u_a)} \sum_{j=1}^{\mathfrak{r}} (\mathsf{alloc}(v^a)_j - \mathsf{edw}_V(v^a)_j) \cdot \frac{2}{\delta+2} \cdot \begin{cases} 2 \text{ or} \\ \mathrm{DCF}_j \end{cases} \\ &- \mathsf{qtc}(u_a)\end{aligned} \tag{5.1}$$

holds. Due to (a), $\forall u_a \in U\colon \mathsf{qtc}(u_a) = q'$ for some $q' \in \mathbb{R}$ holds. Define

$$\mathsf{w}(u_a) := \sum_{v^a \in \mathsf{own}(u_a)} \sum_{j=1}^{\mathfrak{r}} (\mathsf{alloc}(v^a)_j - \mathsf{edw}_V(v^a)_j) \cdot \begin{cases} 2 \text{ or} \\ \mathrm{DCF}_j. \end{cases}$$

Then Equation 5.1 can be rewritten as

$$\mathsf{G}_U^\delta(u_a) = -q' + s + \frac{2}{\delta+2} \cdot \mathsf{w}(u_a) \tag{5.2}$$

Let $A_g^{\delta'}$ be a $\mathsf{G}_g^{\delta'}$-allocation for some $0 \leq \delta' \leq 1$. $A_g^{\delta'}$ is generated such that any two users $u_a, u_c \in U$ have the same $\mathsf{G}^{\delta'}$-value. However, due to the partitioning of VMs to nodes and the request vector of VMs, users may also have a different greediness (cf. the proof Theorem 3.9). Therefore, $\mathsf{G}_U^{\delta'}(u_a) = \mathsf{G}_U^{\delta'}(u_c) + x$, for some $x \in \mathbb{R}$ holds and, thus,

$$-q' + s + \frac{2}{\delta+2} \cdot \mathsf{w}(u_a) = -q' + s + \frac{2}{\delta+2} \cdot \mathsf{w}(u_c) + x. \tag{5.3}$$

Equation 5.3 can be simplified to

$$\mathsf{w}(u_a) = \mathsf{w}(u_c) + x. \tag{5.4}$$

Since function w does not contain δ, the ratio of u_a's and u_c's $\mathsf{G}^{\delta'}$-value is independent of δ. Therefore, $A_g^{\delta'}$ is G^{δ}-max-min fair, for any δ, and thus, $A_g^{\delta'}$ is the G_g^{δ}-allocation for any δ. □

Section 5.2.1 investigates a setup that demonstrates how δ influences the G_g^{δ} allocations, when (b) does not hold. From Theorem 5.1 follows the subsequent corollary.

Corollary 5.2. When all VMs have the same static greediness and Equation 4.2 holds, then the G_l^{δ}-allocation is independent of δ.

5.1.2 Characteristics

DRF is the most prominent approach to MRA fairness in data centers. [30] shows that under the assumptions that (a) all resources are provided by one monolithic node and (b) every user is characterized by a Leontief utility function (the abstraction of users to VMs does not exist) with strictly positive demands for every resource, DRF, *i.e.*, the D-policy, is strategy-prove and the resulting D-allocation is envy-free, Pareto-efficient, and provides sharing incentives (cf. Section 2.5). Under these assumptions the C-policy also achieves these characteristics [38] and Theorem 5.3 shows that under these assumptions also the G^{δ}-policy achieves these characteristics [84].

To emulate the assumptions made by DRF, which does not consider individual VMs or user quotas, Theorem 5.3 assumes that every user $u_a \in U$ has the same quota $q \in \mathbb{R}^{\mathfrak{r}}$ and starts one VM v^a with $\mathsf{virt}(v^a) = q$, *i.e.*, v^a is configured with u_a's entire quota. Therefore, u_a's greediness is equal to v^a's dynamic greediness, as u_a's quota credit and v^a's static greediness cancel each other out (thus, the G^{δ}-policy's scope is irrelevant in the subsequent discussion). To ease the discussion but without loss of generality, Theorem 5.3 assumes that the monolithic node is not overcommitted, wherefore $\mathsf{virt}(v^a) = \mathsf{edw}_V(v^a)$ holds (cf. Section 4.1).

Theorem 5.3. *When (a) a cloud contains only one node n, (b) n is not overcommited, (c) every user $u_a \in U$ has the same quota $q \in \mathbb{R}^{\mathfrak{r}}$, (d) starts one VM v^a with $\mathsf{virt}(v^a) = q$, (e) and each user/VM is characterized by a Leontief utility function with strictly positive demands for every resource, then the G^{δ}-policy is strategy-prove and provides envy freeness, Pareto efficiency, and sharing incentives.*

Proof. The four characteristics are proven separately.

Sharing incentive: An allocation A provides sharing incentives, when A gives each user $u_a \in U$ a utility that is at least as high as when u_a had exclusive access to the resources that correspond to u_a's quota. Assume a G_g^δ-allocation A, for some $0 \leq \delta \leq 1$, does not provide sharing incentives. As every VM receives resources in the ratio of its request vector, a VM v^c must exist that receives strictly less of every resource than v^c's VRs, *i.e.*, $\mathsf{alloc}(v^c) < \mathsf{virt}(v^c)$. Because (a) $\mathsf{alloc}(v^c) < \mathsf{virt}(v^c) = \mathsf{edw}_V(v^c)$, (b) $\frac{2}{2+\delta} > 0$, and (c) at least one resource is saturated, v^c's dynamic greediness is negative. Due to the factors $2 \cdot \frac{2}{2+\delta}$ and $\frac{4}{2+\delta}$, which are applied when ceding or exceeding resources, and since v^c cedes resources, another VM must exist, that has a positive dynamic greediness. Accordingly, the dynamic greediness of VMs is not equal and, thus, the G^δ-values of users not aligned. Then A is not a G_g^δ-allocation.

Pareto-efficiency: DRF makes the assumption that every user is characterized by a Leontief utility function, wherefore, this assumption is also made here. This assumption combined with the fact that the G_g^δ-allocation saturates at least one resource (cf. Listing 4.1), implies that Pareto-efficiency is trivially achieved (cf. [38] and Section 2.5.3).

Strategy proofness: Assume that lying about v_c's requirement vector, *i.e.*, the vector that specifies the ratio in which resources are allocated to v^c, can increase u^c's utility. Because strictly positive Leontief utility functions are assumed, v^c has to receive more of every resource, in order to increase u^c's utility. Because the G_g-policy provides sharing incentives, v^c receives at least its endowment of at least one resource r, when the true requirement vector is reported. Receiving more of r increases v^c's G-value. Since the G-values are aligned, the G-value increase caused by receiving more of r has to be compensated by receiving less of another resource r'. Thus, in order to receive more of r, u^c has to request/receive less of r'. Then u^c's utility decreases.

Envy-freeness: Assume two VMs $v^a, v^c \in V$ have the same VRs. Furthermore, assume u_a envies v^c's bundle. Then v^c receives strictly more than v^a of at least one resource. Since both VMs have the same G-value, v^c has to receive strictly less than v^a of another resource. However, because the G_g-policy is strategy-prove, u_a has stated the true requirement vector of v^a. Therefore, when v^c receives less of a resource than v^a, u_a cannot envy v^c's bundle. □

5.1.3 Analytical Results

When nodes are not overcommitted and all users are equal, *i.e.*, have the same quota, and deploy an equal amount of that quota (to instantiate VMs), the allocation that results from applying the G_g^δ-policy is independent of δ. Similarly, the G_l^δ-allocation is independent of δ, when nodes are not overcommitted and all VMs have the same static greediness.

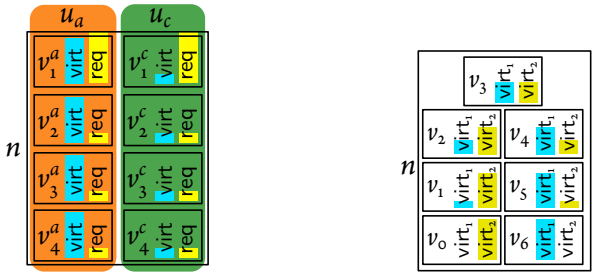
Under the assumption of Leontief utility functions, the G^δ-policy is (a) strategy-prove, (b) envy-free, (c) Pareto-efficient, and (d) provides sharing incentives. However, also the D- and C-policy have these characteristics. The simulative investigation in the next section shows that despite this initial tie between the policies in terms of desirable characteristics the G^δ-policy is superior.

5.2 Simulative Investigation

Different M-policies are applied to different simulation setups and the resulting fairness is compared. Section 5.2.1 and 5.2.2 investigates traits unique to the G^δ-policy. In particular, effects of different δ and the configuration incentive are highlighted by simulations. Subsequently, Section 5.2.3, 5.2.4, and 5.2.5 compare the G^δ-, D-, and C- policy on different simulation setups. Section 5.2.6 discusses starvation limits and their effects and Section 5.2.7 finally summarizes the results of the theoretical investigations.

5.2.1 Setup 1: Configuration Strategies and δ

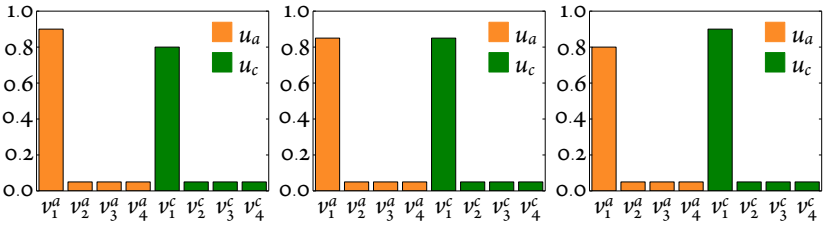
This setup compares two strategies to configure VMs and demonstrates the effects of different values of δ and is illustrated in Figure 5.1a. One node n provides one resource r in quantity 2, *i.e.*, $\mathsf{phy}(n) = 2$. Node n is shared by two users u_a and u_c, who have a quota of 1, *i.e.*, $\mathsf{quota}(u_a) = \mathsf{quota}(u_c) = 1$. Both users want to execute a workload that requires four VMs. Three of these VMs will each require 0.05 of r and one VM will require as much of r as is available. When configuring the VMs, the users know that three VMs will require 0.05 but they do not know, which VM will require infinite resources. Therefore, both users instantiate four VMs and choose different strategies to configure these VMs. Because one VM will require infinite re-



(a) Setup 1

(b) Setup 2 for $q = 6$. Every VM belongs to a different user and requests an infinite amount of resources in the ratio as v_4's VRs

Figure 5.1: Illustrations of Setup 1 and Setup 2



(a) $\delta = 0$ **(b)** $\delta = 0.5$ **(c)** $\delta = 1$

Figure 5.2: G_g^δ-allocations for Setup 1. VMs are on the x-axis and resource amounts are on the y-axis

sources, u_a chooses the strategy to partition the entire quota to the four VMs $v_1^a, v_2^a, v_3^a, v_4^a$, *i.e.*, $\mathsf{virt}(v_x^a) = 0.25$ for $1 \leq x \leq 4$. As three of the four VMs will only require an amount of 0.05 of r, u_c chooses the strategy to configure each VM $v_1^c, v_2^c, v_3^c, v_4^c$ with that amount of VRs, *i.e.*, $\mathsf{virt}(v_x^c) = 0.05$ for $1 \leq x \leq 4$.

During runtime, the first VM of both users happens to be the *busy* VM, *i.e.*, attempts to utilize an infinite amount of r, while the six other VMs are *idle*, *i.e.*, attempt to utilize an amount of 0.05 of r. Figure 5.2 (backed by full numerical details as of Table 5.1) illustrates the G_g^0-, $G_g^{0.5}$-, and G_g^1-allocations. The figure shows that all idle VMs receive all requested resources. However, bundles of the two busy VMs differ depending on δ.

Figure 5.2a depicts that for $\delta = 0$, v_1^a receives more resources than v_1^c. The reason is that $\mathrm{DCF} = 1 = \frac{2}{2+\delta}$ and, thus, the over-configuration of v_2^a, v_3^a, v_4^a is not penalized (cf. Equation 3.6). Accordingly, Table 5.1 shows that the idle VMs of both users have the same G°-values. However, $\mathsf{virt}(v_1^a) = 5 \cdot \mathsf{virt}(v_1^c)$, wherefore, v_1^a receives more resources than v_1^c. When δ increases, the over-configuration of u_a's idle VMs is penalized stronger and the under-configuration of u_c's busy VM is penalized less. Accordingly, Figure 5.2b and Table 5.1 display that $\delta = 0.5$ is the tipping point, at which VMs of both users receive the same amount of resources. Table 5.1 depitcs that the G^δ-values of u_a's idle VMs are maximized for $\delta = 1$. Accordingly, v_1^a receives least resources for $\delta = 1$ and v_1^c most, in order to align both users' G^1-values.

5.2.1.1 Generalization

Setup 1 can be altered in two dimension: the number of idle VMs and the amount of resources idle VMs request (while adapting the VRs of u_c's VMs accordingly). Let $\mathfrak{s} \in \mathbb{N}_{\geq 1}$ be the number of VMs that each user owns and $0 \leq \mathfrak{t} \leq 1$, such that $\mathfrak{t}/\mathfrak{s}$ is the resource utilization of all idle VMs and the VRs of u_c's VMs. For example, in the scenario discussed above $\mathfrak{s} = 4$ and $\mathfrak{t} = 0.2$. Under these assumptions

$$\mathsf{G}_U^\delta(u_a) = \frac{4}{\delta+2}\left(\mathsf{alloc}(v_1^a) - \frac{1}{\mathfrak{s}}\right) - (\mathfrak{s}-1)\cdot\frac{2}{\delta+2}\left(\frac{1}{\mathfrak{s}} - \frac{\mathfrak{t}}{\mathfrak{s}}\right), \quad (5.5)$$

where the minuend is $\mathsf{G}_V^\delta(v_1^a)$ and the subtrahend is the δ-greediness of the $\mathfrak{s}-1$ idle VMs of u_a. Similarly

$$\mathsf{G}_U^\delta(u_b) = \frac{4}{\delta+2}\left(\mathsf{alloc}(v_1^b) - \frac{\mathfrak{t}}{\mathfrak{s}}\right) + \mathfrak{t} - 1, \quad (5.6)$$

where the first summand is $\mathsf{G}_V^\delta(v_1^b)$. Since the greediness of u_b's idle VMs is 0, this greediness does not appear in Equation 5.6. Summand $\mathfrak{t}-1$ is the remainder of u_b's quota credit. The G_g^δ-policy determines $\mathsf{alloc}(v_1^a)$ and $\mathsf{alloc}(v_1^b)$, such that u_a and u_b have the same G^δ-value, *i.e.*,

$$\mathsf{G}_U^\delta(u_a) = \mathsf{G}_U^\delta(u_b). \quad (5.7)$$

Table 5.1: G^{δ}-values for the G_g^{δ}-allocations illustrated in Figure 5.2

		$\delta = 0$		$\delta = 0.5$		$\delta = 1$		$\delta \in \{0, 0.5, 1\}$	
		v_1^a	v_1^c	v_1^a	v_1^c	v_1^a	v_1^c	$v_{2,3,4}^a$	$v_{2,3,4}^c$
G_g^{δ}-allocation		0.90	0.80	0.85	0.85	0.80	0.90	0.05	0.05
Value	G^0	1.55	1.55	1.45	1.65	1.35	1.75	0.05	0.05
	$G^{0.5}$	1.29	1.25	1.21	1.33	1.13	1.41	0.09	0.05
	G^1	1.12	1.05	1.05	1.12	0.98	1.18	0.12	0.05
		u_a	u_c	u_a	u_c	u_a	u_c		
G_g^{δ}-allocation		1.05	0.95	1.00	1.00	0.95	1.05		
Value	G^0	0.70	0.70	0.60	0.80	0.50	0.90		
	$G^{0.5}$	0.56	0.40	0.48	0.48	0.40	0.56		
	G^1	0.47	0.20	0.40	0.27	0.33	0.33		

A second constraint is that all resources have to be allocated, *i.e.*,

$$2 - 2 \cdot (\mathfrak{s} - 1) \cdot \frac{\mathfrak{t}}{\mathfrak{s}} - \mathsf{alloc}(v_1^a) - \mathsf{alloc}(v_1^b) = 0 \tag{5.8}$$

must hold, where 2 is the overall resource supply and $2 \cdot (\mathfrak{s} - 1) \cdot \frac{\mathfrak{t}}{\mathfrak{s}}$ are the resources allocated to the $2 \cdot (\mathfrak{s} - 1)$ idle VMs. Resolving Equations 5.7 and 5.8 results in

$$\mathsf{alloc}(v_1^a) = 1 - \mathfrak{t} + \frac{(t-1) \cdot \delta + \frac{6 \cdot \mathfrak{t} + 2}{\mathfrak{s}}}{8} \tag{5.9}$$

and

$$\mathsf{alloc}(v_1^c) = 1 - \mathfrak{t} + \frac{(1-\mathfrak{t}) \cdot \delta + \frac{10 \cdot \mathfrak{t} - 2}{\mathfrak{s}}}{8}. \tag{5.10}$$

Calculating the difference

$$\mathsf{alloc}(v_1^a) - \mathsf{alloc}(v_1^c) = \frac{(2 - \mathfrak{s} \cdot \delta) \cdot (1 - \mathfrak{t})}{4 \cdot \mathfrak{s}}, \tag{5.11}$$

reveals that wether u_a's or u_c's strategy is superior depends on a combination of δ and $\mathfrak{s}$, whereat increasing $\mathfrak{s}$ and δ make u_c's strategy preferable. Notably, $\mathfrak{t}$ only changes the difference in the G_g^δ-allocations but does not influence, which strategy is superior. Furthermore, for $\delta = 0$ the first factor of the dividend is always positive, which reflects that u_a's strategy is always superior, as over-configuring VMs is not penalized.

5.2.2 Setup 2: Incentives

This setup shows how the G-policy provides incentives to users to configure VMs correctly and is illustrated in Figure 5.1b. Let $\mathfrak{q} \in \mathbb{N}_{>0}$. There is only one node n. n hosts $\mathfrak{q} + 1$ VMs $v_0, v_1, v_2, \ldots, v_\mathfrak{q}$ and provides two resources r_1 and r_2. Every VM requests an infinite amount of resources in ratio 2:1, *i.e.*, every VM requests twice as much of r_1 than of r_2. The VRs of VM v_j are $\mathsf{virt}(v_i) = (i, \mathfrak{q} - i)$ and $\mathsf{phy}(n) = (\frac{\mathfrak{q}^2 - \mathfrak{q}}{2}, \frac{\mathfrak{q}^2 - \mathfrak{q}}{2}) = \sum_{i=0}^{\mathfrak{q}} \mathsf{virt}(v_i) = \sum_{i=0}^{\mathfrak{q}} \mathsf{edw}_V(v_i)$, *i.e.*, the VMs' endowments completely partition n's resources. Each VM belongs to a different user. All users have the same quota, wherefore the results presented below are independent of the

amount of this quota. Section 5.1.1 shows why the results are independent of δ and the scope (local/global).

Figure 5.3 illustrates the G-allocation for $\mathfrak{q} \in \{3, 6, 9\}$, as these are the smallest values for $\mathfrak{q}$ for which a VM exists that is perfectly configured, *i.e.*, whose VRs have ratio 2:1 (just as the request ratio). While δ influences the G^{δ}-values of VMs, it does not influence the G^{δ}-allocation. For example, for $\mathfrak{q} = 6$, $\mathsf{G}^{0}_{V}(v_i) = 0.920$, $\mathsf{G}^{0.25}_{V}(v_i) = 0.818$, $\mathsf{G}^{0.5}_{V}(v_i) = 0.736$, $\mathsf{G}^{0.75}_{V}(v_i) = 0.669$, $\mathsf{G}^{1}_{V}(v_i) = 0.614$ for all $v_i \in V$ holds. This reflects that for smaller δ, deviating from the endowment increases the greediness stronger.

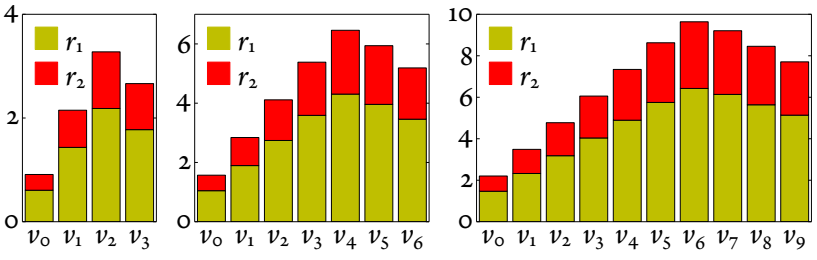
Although all VMs have the same sum of VRs, Figure 5.3 shows that VMs receive different amounts of resources, because the ratios of their VRs are different. In particular, the better a VM's VR ratio is aligned with the actual resource requirement, the more resources the VM receives. Accordingly, $\mathsf{virt}(v_0) = (0, \mathfrak{q})$ is the worst configuration to request resources in ratio $(2, 1)$ and, thus, v_0 receives the smallest bundle for all $\mathfrak{q}$. In contrast, VM $v_{2\mathfrak{q}/3}$'s VRs have the perfect ratio, wherefore v_2, v_4 and v_6 receive the most resources for $\mathfrak{q} = 3$, $\mathfrak{q} = 6$, and $\mathfrak{q} = 9$, respectively. Although v_0 and $v_{\mathfrak{q}}$ have one VR configured with 0, $v_{\mathfrak{q}}$ receives significantly more resources than v_0, because it has a high endowment to the stronger requested resource.

5.2.3 Setup 3: VM and User-based Fairness

This setup demonstrates the importance of enforcing fairness on a per-user and not per-VM basis and is illustrated in Figure 5.4a. A single node $n_x \in N$ hosts different numbers of VMs of two users $u_a, u_c \in U$ that compete for one resource r. In particular, let u_a own two VMs $\dot{v}^a_x, \ddot{v}^a_x$ and u_c own one VM v^c_x. Six infinitely devisable units are available of r. Both users have a quota of three units to r and all VMs have an endowment of 1. Normalization as described in Section 3.3.1.3 is applied, wherefore one unit of r has a value of $2/6 = 0.\bar{3}$.

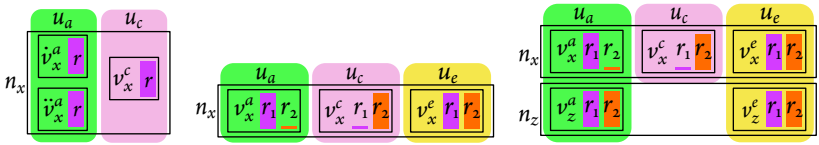
5.2.3.1 Local Policies

All M_l-policies treat all VMs equally, as they, by definition, do not consider the owner of VMs. Therefore, all M_l-policies allocate all VMs two units of r,



(a) $q = 3$ **(b)** $q = 6$ **(c)** $q = 9$

Figure 5.3: G-allocations for Setup 2 for different numbers of VMs (q). VMs are on the x-axis and resource amounts are on the y-axis



(a) Setup 3 **(b)** Setup 4 **(c)** Setup 5

Figure 5.4: Illustrations of Setup 3, Setup 4, and Setup 5

i.e., all M_l-allocations are the same. Consequently, u_a receives twice as much resources as u_c.

Table 5.2 shows the M-values and M-scores for this allocation row-wise for different M. Since there is only one resource, the dominant share of a bundle is equal to the amount of the only resource in the bundle. Therefore, the C- and D-value of all bundles is equal. The G^δ-values depend on δ. The reason is that VMs receive two units of r, while their endowment is 1. This exceeding of the endowment is accounted for differently depending on δ. As the G^δ-value of users contains the sum of G^δ-values of their VMs, also the G^δ-values of users are different.

Table 5.2 depicts that all M-scores are 0.9. The reason is that for any metric M the M-value of u_a is twice the M-value of u_c, when input into Jain's index to calculate the M-score (cf. Equation 4.8). For the C- and D-metric, this follows directly from the C- and D-values in the table. The G^δ-values shown

Table 5.2: M-values and M-scores for the local allocation of Setup 3

	M-value					M-score
	$\dot{v}_x^a$	$\ddot{v}_x^a$	v_x^c	u_a	u_c	
C/D	$0.\overline{6}$	$0.\overline{6}$	$0.\overline{6}$	$1.\overline{3}$	$0.\overline{6}$	0.9
G^0	1	1	1	1	0	0.9
$G^{0.5}$	$0.8\overline{6}$	$0.8\overline{6}$	$0.8\overline{6}$	$0.7\overline{3}$	$-0.1\overline{3}$	0.9
G^1	$0.\overline{7}$	$0.\overline{7}$	$0.\overline{7}$	$0.\overline{5}$	$-0.\overline{2}$	0.9

in the table for u_a and u_c do not have this 2:1 ratio. However, as explained in Section 4.1.4, these G^δ-values are increased by 1 before input to Jain's index.

5.2.3.2 Global Policies

Table 5.3 lists the M_g-allocations on a per-user basis and the corresponding M-scores. Contrary to Table 5.2, the M-values are not shown. Also it is not listed, what individual VMs receive: as u_a's VMs are identical, they receive the same amount of r, *i.e.*, 50% of what u_a receives, while u_c's entire amount is allocated to u_c's only VM v_x^c. Different M_g-allocations are depicted row-wise, while the corresponding M-scores are presented in the four rightmost columns. For example, the G^0-score of the D_g- or C_g-allocation is 0.987. The table indicates that the smaller δ is, the less the G_g^δ-policy allocates u_c. The reason is that a smaller δ penalizes under-configuring VMs more strongly. As u_c runs two VMs, u_a has configured more resources and, therefore, under-configured less than u_c, who only has configured one VM. Also the G^0-scores reflect that G^0 disfavors u_c, as, the larger u_c's bundle, the lower the corresponding G^0-score. However, in general, all M_g-allocations are similar, wherefore, all M-scores are high. The diagonal of 1s in the right half of the table shows that, for a given metric M, the M_g-allocation also has the highest possible M-score. The M_l-allocation described in Table 5.2 does not achieve the maximal M-score for any metric M, as this allocation is local and, therefore, achieves fairness on a per-VM basis, while the M-scores evaluate fairness on a per-user basis.

Table 5.3: Global M-allocations and corresponding M-scores for Setup 3. Values beginning with 0.9 are rounded

M-Allocation			M-Score			
	u_a	u_c	C/D	G^0	$G^{0.5}$	G^1
C/D	3	3	1	0.987	0.994	0.997
G^0	3.25	2.75	0.993	1	0.999	0.997
$G^{0.5}$	3.1875	2.8125	0.996	0.999	1	0.999
G^1	3.125	2.875	0.998	0.997	0.999	1

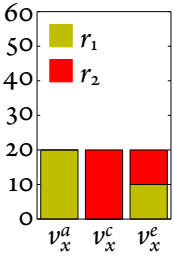
5.2.4 Setup 4: Exchange of Resources on the Same Node

In this setup (cf. Figure 5.4b) there are two resources r_1 and r_2, one node n_x, with $\mathsf{phy}(n_x) = (30, 30)$, and three users u_a, u_c, u_e, which all have quota $(20, 20)$ and own VM v_x^a, v_x^c, v_x^e, respectively. All VMs are scheduled on n_x and have an endowment of $(10, 10)$. The VMs' demands are $\mathsf{req}(v_x^a) = (30, 0)$, $\mathsf{req}(v_x^c) = (0, 30)$, and $\mathsf{req}(v_x^e) = (30, 30)$, *i.e.*, the VM of customer u_a attempts to fully utilize r_1, the VM of customer u_c attempts to fully utilize r_2, and the VM of u_e attempts to fully utilize both resources.

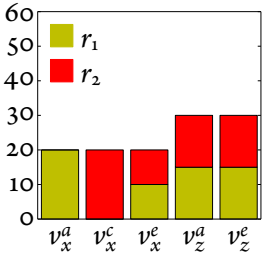
Since all users have the same quota and only operate one VM, allocations are independent of the scope, which is therefore omitted in the discussion. Section 5.1.1 explains, why the G^δ-allocations are the same for all δ. Accordingly, also parameter δ is omitted in the discussion of this section. Figures 5.5a, 5.5d, and 5.5g illustrate the local and global C-allocation, D-allocation, and G-allocation (as just discussed, δ is omitted), respectively.

These figures show that the C-policy allocates all VMs the same sum of resources, while the D-policy allocates v_x^e twice the amount of resources, as it only considers one dominant share (cf. Section 2.6.1.4.2). The G-policy takes the middle way, as it slightly favours v_x^e. The reason is that all VMs have the same symmetric configuration of VRs, while only v_x^e's resource demand is symmetric and v_x^a and v_x^c request only one resource. Therefore, v_x^e is better configured and gets an accordingly larger share of n_x's resources.

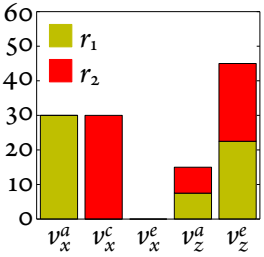
Table 5.4 describes the allocations in Figure 5.5a, 5.5d, and 5.5g numerically and lists the respective M-scores. Notably, the G^δ-scores of the C- and



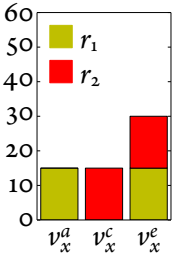
(a) C-allocation for Setup 4



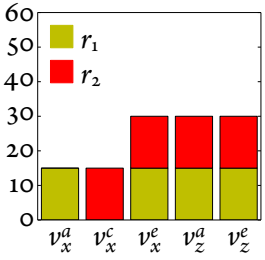
(b) C_l-allocation for Setup 5



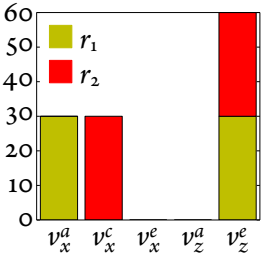
(c) C_g-allocation for Setup 5



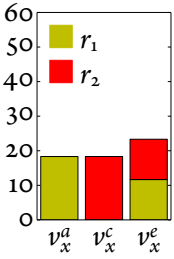
(d) D-allocation for Setup 4



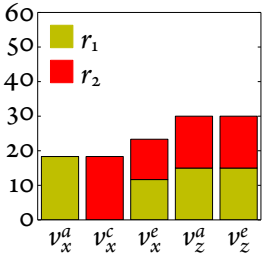
(e) D_l-allocation for Setup 5



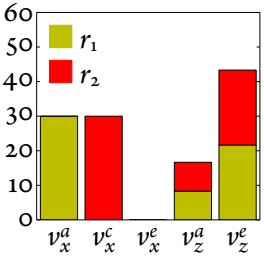
(f) D_g-allocation for Setup 5



(g) G-allocation for Setup 4



(h) G_l-allocation for Setup 5



(i) G_g-allocation for Setup 5

Figure 5.5: Illustration of the various M-allocations for Setup 4 and Setup 5. VMs are on the x-axis and resource amounts are on the y-axis

Table 5.4: M-allocations and corresponding M-scores of Setup 4. Values beginning with 0.9 are rounded

	M-allocation			M-Score				
	u_a	u_c	u_e	C	D	G^0	$G^{0.5}$	G^1
C	20,0	0,20	10,10	1	9.926	0.969	0.978	0.984
D	15,0	0,15	15,15	$0.\overline{8}$	1	$0.\overline{8}$	0.919	0.937
G	$18.\overline{3}$, 0	0, $18.\overline{3}$	$11.\overline{6}$, $11.\overline{6}$	0.986	0.963	1	1	1

Table 5.5: M-values of the three different M-allocations of Setup 4

	C-allocation			D-allocation			G-allocation		
	u_a	u_c	u_e	u_a	u_c	u_e	u_a	u_c	u_e
C	1	1	1	0.75	0.75	1.5	$0.91\overline{6}$	$0.91\overline{6}$	$1.1\overline{6}$
D	1	1	0.5	0.75	0.75	0.75	$0.91\overline{6}$	$0.91\overline{6}$	$0.58\overline{3}$
G^0	-0.5	-0.5	-1	-1	-1	0	$-0.\overline{6}$	$-0.\overline{6}$	$-0.\overline{6}$
$G^{0.5}$	-0.6	-0.6	-1	-1	-1	-0.2	$-0.7\overline{3}$	$-0.7\overline{3}$	$-0.7\overline{3}$
G^1	$-0.\overline{6}$	$-0.\overline{6}$	-1	-1	-1	$0.\overline{3}$	$-0.\overline{7}$	$-0.\overline{7}$	$-0.\overline{7}$

D-allocation differ slightly depending on δ. The reason for this can be found in Table 5.5, which depicts the users' M-values. The table shows that for the C- and D-allocation some users' G^δ-values slightly increase with decreasing δ, wherefore, also the G^δ-scores in Table 5.4 change depending on δ. For example, the G^0-values for the C-allocation are -0.5, -0.5, and -1, for user u_a, u_c, and u_e, respectively, while the the G^1-values are $-0.\overline{6}$, $-0.\overline{6}$, and -1. This leads to G^δ-scores that change depending on δ.

5.2.5 Setup 5: Cross Host Alignment

Setup 5 extends Setup 4 by adding node n_z that has the same capacities as n_x and that hosts two additional VMs (cf. Figure 5.4c). The additional VMs, v_z^a and v_z^e, are owned by customer u_a and u_e, respectively. v_z^a and v_z^e have an endowment of $(10, 10)$ and attempt to fully utilize the node, *i.e.*, $\mathsf{req}(v_z^a) = \mathsf{req}(v_z^e) = (30, 30)$.

Contrary to Setup 4, in Setup 5 the scope makes a difference, as users now run different numbers of VMs. Just as in Setup 4, the G_g^δ-allocations are independent of δ, for the following reason: u_c only runs one VM, wherefore the share of this VM v_x^c has to be maximised, in order to align the G^δ-value of users as much as possible for any δ. Accordingly, v_x^c receives all of n_x's supply of r_2, which implies that no resources can be allocated to v_x^e, as v_x^e also runs on n_x and requires r_2. v_x^a, which is the third VM on n_x, only requires r_1 and therefore receives all of r_1, since the allocation has to be PE. Thus, the allocation on n_x is "fixed" and only the allocation on n_z can potentially vary for different δ. However, n_z hosts VMs of users, who deploy the same percentage of their quota. Also these users have the same quota and Equation 4.2 holds. Therefore, an argumentation analogous to the proof of Theorem 5.1 proves that δ does not change the G_g^δ-allocation on n_z and, thus, all G_g^δ-allocations are equal. A simpler argumentation shows that also the G_l^δ-allocations are independent of δ as well.

Figure 5.5b and 5.5c show the C_l- and C_g-allocation, respectively, Figure 5.5e and 5.5f illustrate the D_l- and D_g-allocation, respectively, and Figure 5.5h and 5.5i depict the G_l- and G_g-allocation, respectively.

The local allocations for Setup 5 are unspectacular: On node n_z all allocations are the same and the allocations on n_x are the same as for Setup 4.

As every M_g-policy attempts to equalize the M-values users receive, all M_g-policies maximize the share of v_x^c and, therefore, starve v_x^e. Accordingly, the allocation on n_x is equal for all M_g-allocations. On n_z the global allocations differ. In particular, every M_g-policy distributes n_x's resources such that u_a and u_e receive the same M-value. Notably, the D-policy is able to equalize the D-value of all three users. Just as the G-allocation lies between the D- and C-allocation for Setup 4, the G_g-allocation on n_z lies between the D_g- and C_g-allocation for Setup 5, as the G_g-policy on n_z allocates v_z^a more than the C_g-policy but less than the D_g-policy.

Table 5.6 describes all allocations of Setup 5 numerically and gives the respective M-scores. The table displays that only the D_g-policy is able to achieve perfect fairness by its own definition (the D-score). All other M-policies achieve a lower M-score, as u_c only has one VM and, therefore, can only receive limited resources. Every M_g-policy outperforms its local coun-

terpart in terms of M-score. The reason is that the local policies achieve fairness on a per-VM basis, while global policies achieve fairness on a per-user basis and also the M-score is determined on a per-user basis. While the D-score of the D_g-allocation is 1, the M-score score for $\mathsf{M} \in \{\mathsf{C}, \mathsf{G}^0, \mathsf{G}^{0.5}, \mathsf{G}^1, \}$ is rather low. In contrast, G-scores of the C_g-allocation are high and the C-score of the G_g-allocation is high as well.

5.2.6 Starvation Limits

Setup 5 demonstrated that global policies may starve VMs, *i.e.,* allocate the VMs zero resources. This is not desirable in reality, as it makes VMs useless. Therefore, the simulations were extended by starvation limits, *i.e.,* minimum guaranteed amounts of resources that VMs receive, even though they would have to receive less in order to enforce max-min fairness among the M-values of users. Two different types of starvation limits were implemented and are discussed in Section 5.2.6.1 and 5.2.6.2.

5.2.6.1 Static

Static starvation limits are defined by a simulation wide starvation factor $0 \leq \mathscr{S} \leq 1$ that specifies the fraction VMs are guaranteed to receive of their endowment. In particular, as resources are allocated in the same ratio as VMs request them, $v \in V$ is guaranteed bundle

$$\mathscr{S} \cdot \mathsf{req}(v) \cdot \min_{r_i \in R} \begin{cases} \frac{\mathsf{edw}_V(v)_i}{\mathsf{req}(v)_i} & \text{if } \mathsf{req}(v)_i > 0 \\ \infty & \text{else.} \end{cases} \tag{5.12}$$

5.2.6.2 Dynamic

Dynamic starvation limits define a starvation factor for every user based on the users' M-value. In particular, the higher the M-value of a user, the lower the starvation factor of the user's VMs.

The starvation factor has to be ≤ 1, as the endowment specifies how many resources are budgeted for a VM, and a starvation factor greater 1 means that more resources are guaranteed to a VM than are budgeted for the VM. Furthermore, the starvation factor has to be ≥ 0, as 0 means that no guarantees

Table 5.6: M-allocations and corresponding M-scores of Setup 5. Values beginning with 0.9 are rounded

		M-allocation			M-Score				
		u_a	u_c	u_e	C	D	G^0	$G^{0.5}$	G^1
Local	C	35,15	0,20	25,25	$0.\overline{8}$	0.948	0.901	0.911	0.913
	D	30,15	0,15	30,30	0.821	0.926	0.821	0.837	0.850
	G	$33.\overline{3}$,15	0,$18.\overline{3}$	$26.\overline{6}$,$26.\overline{6}$	0.870	0.948	$0.\overline{8}$	0.894	0.898
Global	C	37.5,7.5	0,30	22.5,22.5	0.970	0.960	0.981	0.977	0.973
	D	30,0	0,30	30,30	$0.\overline{8}$	1	$0.\overline{8}$	0.901	0.910
	G	$38.\overline{3}$,$8.\overline{3}$	0,30	$21.\overline{6}$,$21.\overline{6}$	0.969	0.951	0.984	0.979	0.975

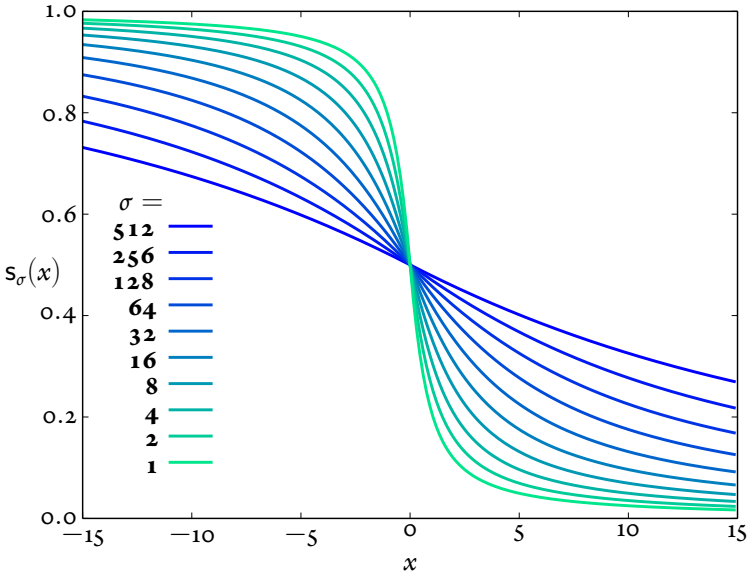


Figure 5.6: Exemplary starvation function s_σ plotted for different σ

are made and there cannot be less than no guarantees. Thus, the starvation factor for a user has to be defined by a function $s\colon \mathbb{R} \to [0,1]$ that monotonically decreases. Also the function should be continuous such that starvation factors gradually change.

Function

$$s_\sigma(x) \mapsto 0.5 + 0.5 \cdot \begin{cases} -h_\sigma(-x) & \text{if } x < 0 \\ h_\sigma(\ \ x) & \text{else,} \end{cases} \tag{5.13}$$

where

$$h_\sigma(x) \mapsto \sqrt{\frac{x^2+\sigma}{\sigma}} - \frac{x}{\sqrt{\sigma}} \tag{5.14}$$

is (a) parameterized by $\sigma > 0$, which controls how fast $s_\sigma(x)$ converges, (b) plotted in Figure 5.6 for different σ, and (c) an exemplary starvation function that meets these demands.

5.2.6.3 Starvation Limit Effects

Simulations that incorporate starvation limits did not yield additional insights about the different metrics. Therefore, those simulations are not discussed in detail. However, two important findings about starvation limits are the following.

Firstly, when starvation limits are static, M-scores generally decrease with increasing $\mathscr{S}$. The reason is that, the higher $\mathscr{S}$ is the less flexibility policies have to allocate resources, as they have to adhere to higher and, therefore, more constraining starvation limits.

Secondly, by design dynamic starvation limits give more incentive to users to behave economically. The reason is that the higher a user u's M-value, the lower u's VMs' dynamic starvation limits. Therefore, if u utilizes a large amount of resources with some VMs (or, in case of the greediness metric, configures VMs poorly) the resources guaranteed to any of u's VMs decrease. This dependency increases the smaller σ is, wherefore, also the incentive to behave economically increases with decreasing σ. In contrast, static starvation limits always guarantee the same amount of resources to u's VMs, even if some of the user's VMs already utilize large amounts of resources.

5.2.6.4 Using s_σ for Asymptotic PP Mappings

Section 4.2.5 discussed mapping greediness to priorities. Function p is presented in Equation 4.9 and maps an input value g into an output interval based on an input interval. The target value is determined, such that its position in the output interval is relatively the same, as g's position in the input interval. Therefore, a linear dependency between input and output values exists, while it was not proven that such dependency is optimal in practice. In fact, it might be advantageous to use an asymptotic function that converges towards the borders of the output interval. One function to implement such asymptotic behavior is $\mathsf{p}(\mathsf{s}_\sigma(g))$, where $\mathrm{in}_{\min} = 0$ and $\mathrm{in}_{\max} = 1$. An advantage of this nested formula is that increasing g always decreases the priority. In contrast, when g is input to p directly, *i.e.*, without applying s_σ first, the priority does not decrease, when $g \geq \mathrm{in}_{\max}$. Therefore, all users with $g \geq \mathrm{in}_{\max}$ are assigned the same priorities, although their greediness may differ.

5.2.7 Simulative and Theoretical Results

The simulations above complete the theoretical investigations and allow for comparing the three metrics in terms of different aspects. Table 5.7 summarizes these findings.

Setup 3 and Setup 5 prove that it is insufficient to consider each VM separately, when aiming for fairness among users. In particular, Setup 3 is very simple as it only contains one node, two users, and three VMs, but already demonstrates, how local policies may fail to achieve fairness.

Fairness was measured by the M-score for different metrics $\mathsf{M} \in \{\mathsf{C}, \mathsf{D}, \mathsf{G}^0, \mathsf{G}^{0.5}, \mathsf{G}^1, \}$. The C-score compares bundles of users by the L_1 metric, which is the most straight-forward bundle measure. The D-score compares bundles of users by the L_∞ metric and is supported by DRF's wide application in data centers. The G^δ-score is backed by the questionnaire presented in Section 3.2. The questionnaire also identified the L_1 and L_∞ norms as insufficient, wherefore the G^δ-score is the best choice. Furthermore, Setup 4 and Setup 5 show that an allocation that maximizes the G^δ-score lies in between allocations that maximize the C- and D-score.

Setup 3 and Setup 5 indicate that the M-score of the M_g-allocation is always at least as high as the M-score of the M_l-allocation. As the G^δ-score best measures the intuitive fairness of an allocation and is maximized by the G^δ_g-allocation, it is concluded that the G^δ_g-policy achieves the highest overall intuitive fairness. While all three policies (G^δ, C, and D) are SI, PE, SP, EF (cf. Section 5.1.2 and [30, 38]) only the G^δ-policy gives incentive to users to configure their VMs correctly (cf. Section 3.3.4.2 and Setup 2). While $0 \leq \delta \leq 1$ has to be fixed, several setups show that the G^δ-score is similar or even equal, even when δ lies at different ends of the spectrum. Furthermore, G^δ-allocations are often independent of δ (cf. Section 5.1.1 and Setup 2, Setup 4, and Setup 5). This makes the G^δ_g-policy the best choice for any δ.

5.3 Experimental Investigations

Controlling cloud resources through PR allocation was a relatively unexplored research field before this thesis. OpenStack [73] is the de facto stan-

Table 5.7: Comparison of the different metrics

Aspect	Finding	Shown in
Scope	per-user always better than per-VM	Setup 3, Setup 5
Intuitiveness	G^{δ}-score superior to C- and D-score	Section 3.3.4.5
Allocations	G^{δ}-allocation lies often between C- and D-allocation	Setup 4, Setup 5
SI, PE, SP, EF	Provided by G^{δ}-, C-, and D-policy	Section 5.1.2, [30, 38]
Incentives	only provided by G^{δ}-policy	Section 3.3.4.2, Setup 2
G^{δ}-policy's δ parameter	irrelevant in several cases	Section 5.1.1, Setup 2, Setup 4, Setup 5

dard for private clouds but out-of-the-box does not control cloud resources by these means. Therefore, Section 4.2 developed an OpenStack extension termed the Fairness Service (FS) that adapts VMs' Proportional Priorities (PP) to enforce fairness according to the definitions established in Section 3.3.3. The FS is evaluated in this section to prove that this novel way to enforce cloud fairness is technically feasible. In particular, Section 5.3.1 evaluates the FS's performance overhead by the CPU time the FS consumes and Section 5.3.2 shows hows this overhead can be decreased. Section 5.3.3 proves that the FS successfully changes the allocation to running VMs by changing their PPs to access PRs based on the PRs their owners overall utilize.

The evaluation environment was set up according to the OpenStack installation guide for Ubuntu 14.04 [71]. All compute nodes are equipped with a 3 GHz dual-core Intel Xeon E3113 CPU, 4 GB RAM, a 150 MiB/s hard-drive, and a 1 Gbit/s network connection.

The efficiency/utilization of the FS is not evaluated, because the FS deploys PPs. These PPs ensure that no PR is idle, if desired by a consumer (cf. Section 2.2.1.1), and, therefore, guarantee high utilization. Furthermore, the FS either increases or decreases the PPs of a VM on all PRs. Thus, no adverse ratios of PPs on different PRs occur.

5.3.1 CPU Overhead

The CPU overhead is evaluated depending on (a) the number of VMs, (b) whether or not VMs are loaded, and (c) the length of the update interval. The evaluation was performed for a single node, as the message exchange among nodes can be implemented, such that the number of messages a node receives and sends is a small constant independent of the overall number of nodes (cf. Section 5.4). Therefore, the number of nodes does not influence the CPU overhead.

Accordingly, a *performance experiment* is defined by the triple $(\beta, \lambda, \mu) \in \mathbb{N}_{\geq 1} \times \{T, F\} \times \mathbb{N} \cup \{\infty\}$. β defines the number of VMs that are hosted by the node in the experiment. The Boolean variable λ specifies whether these VMs are loaded, whereat load is simulated by `stress` [4]. μ determines the length of the update interval. If $\mu = \infty$, the FS is deactivated. Each performance experiment runs for four minutes. The FS's CPU overhead is measured by the CPU time that is utilized by OpenStack processes, when the FS is running ($\mu < \infty$) or not running ($\mu = \infty$). In particular, the OpenStack processes to be monitored are the `nova-compute`, `nova-network`, `nova-api-metadata`, and `nova-fairness` services.

Figure 5.7a and 5.8a show the sum of the CPU time consumed by these processes for different numbers of VMs and update intervals. The figures demonstrate that the FS significantly increases the CPU time that is utilized by these processes.

Controlling network access was identified as main cause for the significant CPU overhead. In particular, as outlined in Section 4.2.5, the Unix application tc is used to apply network priorities, which implies multiple CLI calls to set priorities. Therefore, Section 5.3.2 shows how the CPU overhead caused by controlling network access can be reduced and Figure 5.7b and 5.8b illustrate how much CPU time is utilized by OpenStack processes, when the

FS does not control the network access. These figures depict that disabling network control reduces the CPU overhead by more than 50% on average.

All figures demonstrate that the CPU overhead increases linearly with the number of VMs and shorter update intervals. Therefore, the FS's CPU overhead can be reduced by selecting longer update intervals. In particular, the CPU overhead caused by higher numbers of VMs can be contained by increasing μ.

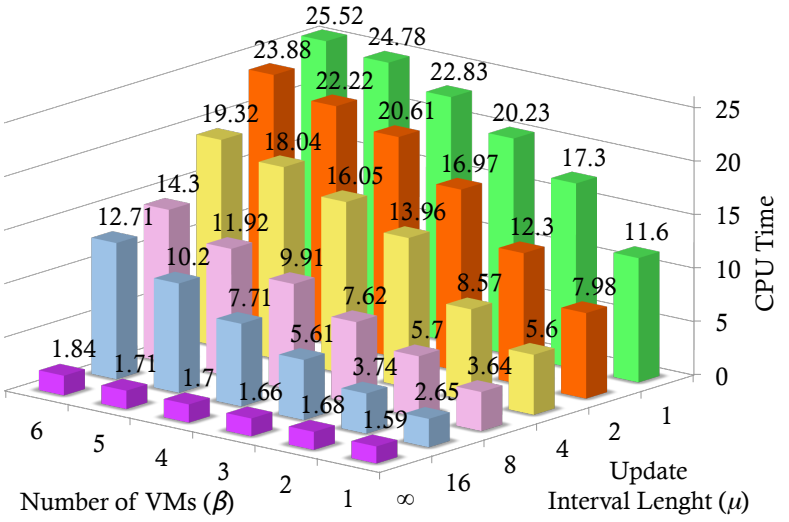
5.3.2 Controlling Network Access

The FS does not use libvirt to control network access, as libvirt only offers hard limits for this purpose. Instead the evaluated prototype of the FS uses tc and the Hierarchical Token Bucket (HTB) queuing discipline to control the network access, as it allows controlling network access by PPs [35]. However, as shown above, this significantly increases the CPU overhead. This section (a) identifies a queuing discipline that is better suited than HTB to control network access by PPs and (b) shows how it can be deployed efficiently by the FS. Circles in the subsequent graphs depict outliers.

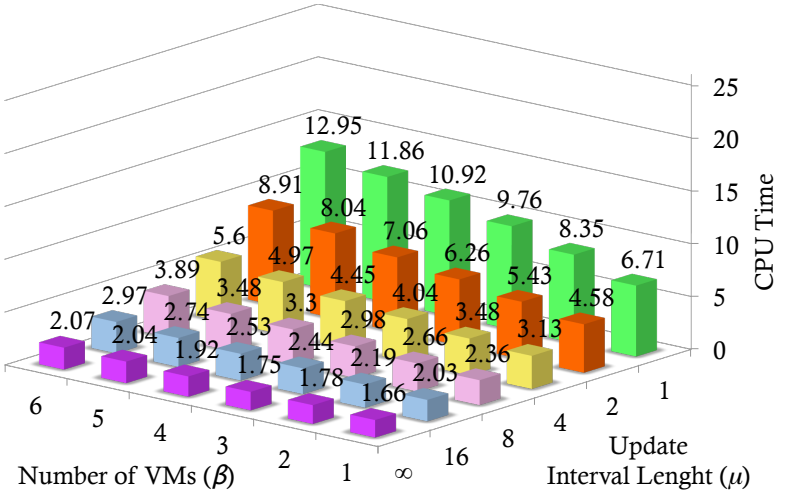
5.3.2.1 Comparison of HTB and HFSC

Literature searches [66, 67] investigated which queuing disciplines allow sharing the network by PPs and identified the Hierarchical Fair Service Curve (HFSC) queuing discipline as potential candidate to replace the Hierarchical Token Bucket (HTB) queuing discipline in the FS. Both queuing disciplines are, thus, compared with respect to (a) how precisely they allocate the network, *i.e.*, the deviation of the actual network allocation from the allocation that is perfect according to the assigned PPs (cf. Section 5.3.2.1.1), (b) how efficiently they utilize the network, *i.e.*, how much percent of the network capacity is utilized in case of congestion (cf. Section 5.3.2.1.2), and (c) how long packets are delayed (cf. Section 5.3.2.1.3).

Subsequently, the term flow is used, as queuing disciplines allocate the network among flows. In the context of the FS, a flow, therefore, is all network traffic produced or received by a VM. The data illustrated in all tables was gathered by 10 iterations of an according experiment.

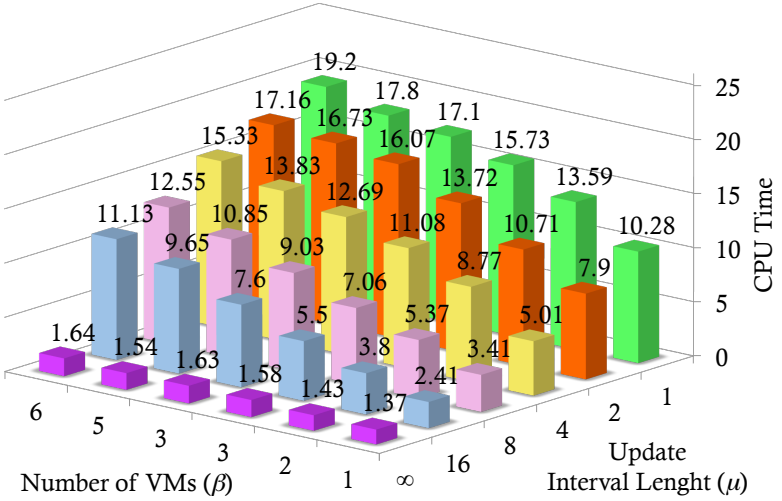


(a) Network is controlled

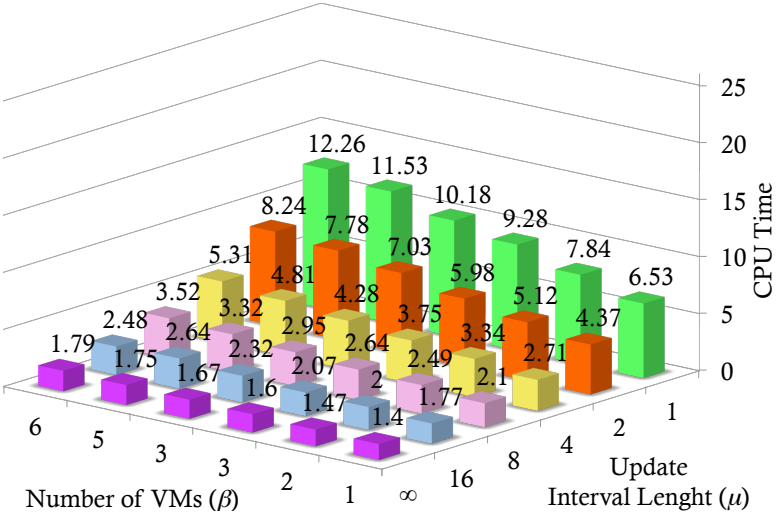


(b) Network is not controlled

Figure 5.7: CPU time consumed by OpenStack services dependent on the number of VMs (β) and the update interval (μ) for $\lambda = F$



(a) Network is controlled



(b) Network is not controlled

Figure 5.8: CPU time consumed by OpenStack services dependent on the number of VMs (β) and the update interval (μ) for $\lambda = T$

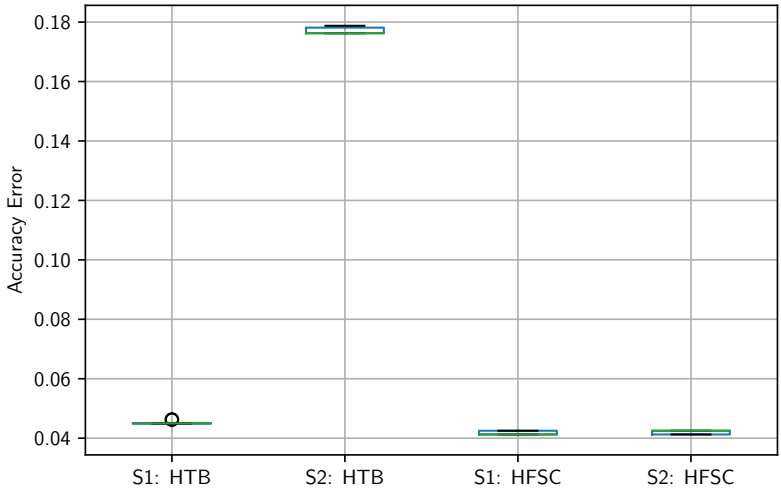


Figure 5.9: Comparison of the allocation accuracy of HTB and HFSC based on two scenarios [67]

5.3.2.1.1 Allocation Accuracy

The allocation accuracy of both queuing disciplines is investigated by two scenarios. In Scenario S1 two flows with different PPs overload the network. In Scenario S2 there are three additional flows and one of them is idle. Therefore, the bandwidth "belonging" to the idle flow has to be distributed among the other flows according to their PPs. Figure 5.9 shows the allocation accuracy by the percentage with which the resulting allocations deviate from the allocation that is perfect according to the PPs. The figure shows that HFSC performs better in both scenarios and HTB has a significantly lower accuracy, when bandwidth of idle flows has to be allocated to other flows.

Figure 5.10 shows that the accuracy of both queuing disciplines does not scale with the number of flows, whereat the scalability of HTB is significantly better.

5.3.2.1.2 Network Utilization

While Figure 5.10 seems to imply that HTB scales better, Figure 5.11 shows that HTB achieves the higher scalability in terms of accuracy at the cost of

utilization. In particular, the figure shows that the utilization of HFSC is significantly higher and stable, while the utilization of HTB varies strongly.

5.3.2.1.3 Packet Waiting Times

HTB results in a mean packet waiting time of 57.5 ms, while HFSC results in a mean packet waiting time of 47 ms. Also the variance of packet waiting times is smaller for HFSC.

5.3.2.2 HFSC Integration into libvirt

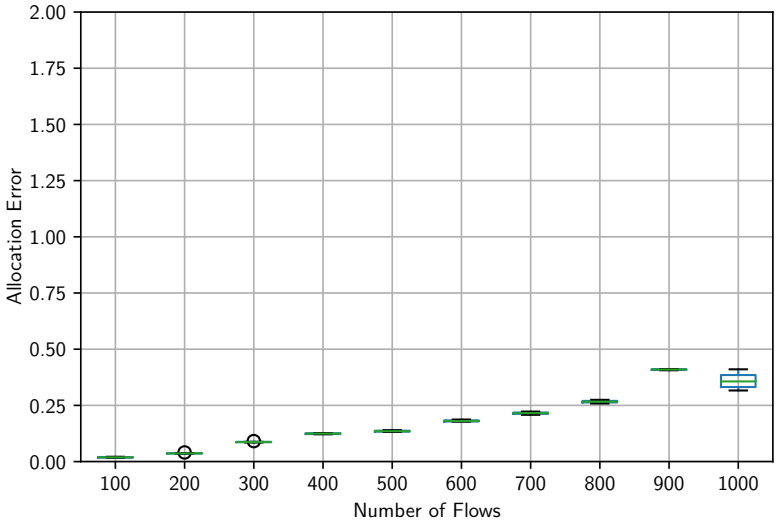
Due to the higher efficiency, lower packet waiting times, and generally more predictable behavior, HFSC is regarded superior to HTB to control network access by PPs. Therefore, libvirt is extended to deploy HFSC to allocate the network by PPs. Integrating the HFSC into libvirt and not the FS directly, makes the resulting implementation useable by all libvirt users independent of the FS. Since the FS uses libvirt to control PRs, the FS is able to deploy HFSC via this extension, without implementation overhead.

The libvirt extension integrates HFSC via hooks, hides most of HFSC's complex parameters, and simply allows controlling the network access by PPs. The PP information required by the extension is conveniently stored in the XML file that libvirt uses to describe a VM. The extension adds a VM that is started to the HFSC hierarchy via a hook (since VMs are only differentiated by their PPs, this hierarchy is flat). Similarly, a VM is removed from the hierarchy by a hook, when the VM is shut down.

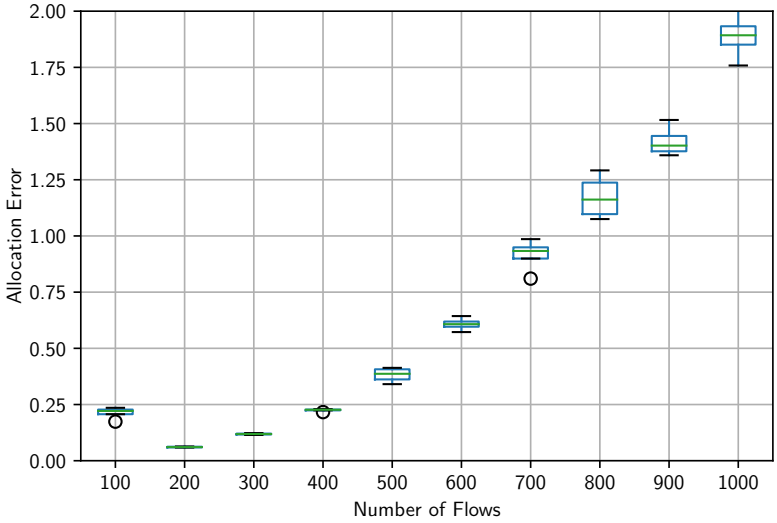
In order to show that this extension allows the FS to efficiently control network access by PPs, the CPU overhead of the extension has to be determined. In particular, the CPU overhead with and without the extension has to be compared, when a VM is started and stopped. Figure 5.12 shows the according results. The negligible CPU overhead of the extension shows that it is possible for the FS to control network access with a low CPU overhead.

5.3.3 Fairness Promotion Among Users

The two *fairness experiments* following show if and how the FS alters VMs' PPs to promote fairness among users. In these fairness experiments the update interval (μ) of the FS is 10 seconds and the FS enforces fairness according

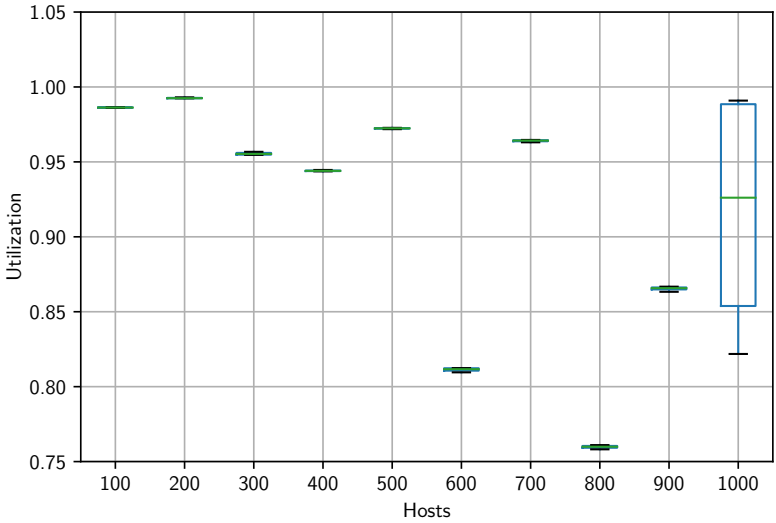


(a) HTB

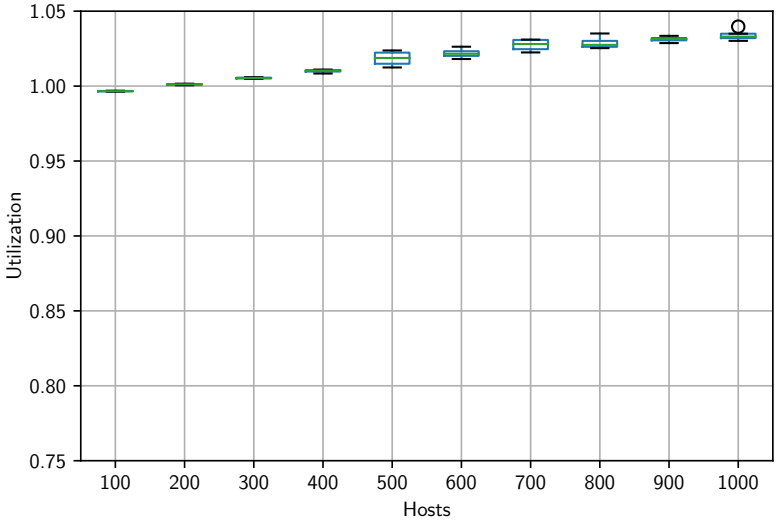


(b) HFSC

Figure 5.10: Allocation accuracy of HTB and HFSC depending on the number of flows [67]



(a) HTB



(b) HFSC

Figure 5.11: Efficiency of HTB and HFSC measured by the utilization of the available bandwidth depending on the number of flows [67]

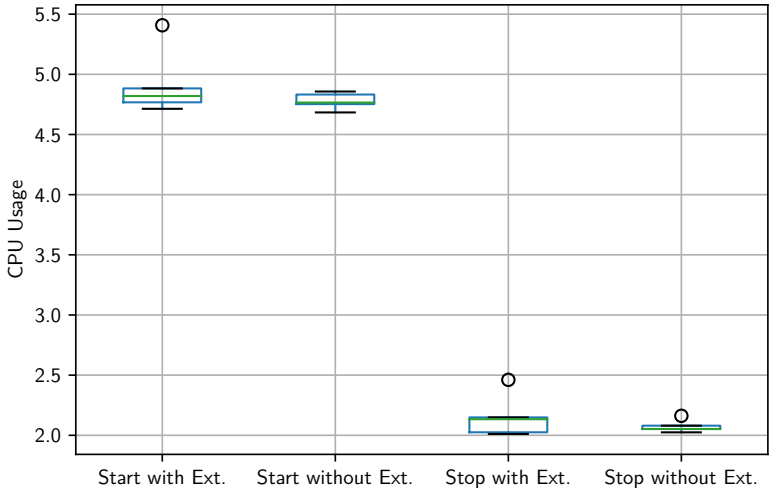


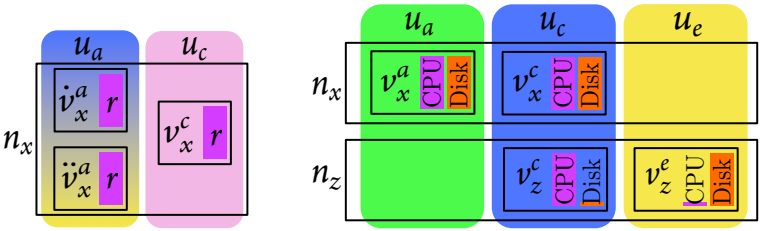
Figure 5.12: CPU time utilized, when starting and stoping a VM with and without the libvirt extension [67]

to Definition 3.6. All evaluations have been conducted over a timespan of 15 minutes. Although the FS is able to monitor and control CPU time, RAM, disk I/O, and network access, only CPU time and disk I/O are considered in these fairness experiments to simplify the discussion. All users in those experiments have the same quota, wherefore the quota credit is the same for all users, and, thus, the size of this quota does not influence results of the experiments.

When the FS is not running, allocations are determined by the Completely Fair Scheduler (CFS) [63], which is the Linux default scheduler and achieves virtually perfect CPU fairness between (VM) processes sharing a node. However, the CFS is oblivious to which cloud user owns a VM process. To the best of the authors' knowledge, the FS is the first implementation that establishes cloud fairness solely by adapting PR allocation to VM processes.

5.3.3.1 Single node, single resource

The constellation of the first fairness experiment (a) is equivalent to Setup 3 that is discussed in Section 5.2.3, (b) illustrated in Figure 5.13a, and (c) in-



(a) One node n_x hosts three VMs $\dot{v}_x^a$, $\ddot{v}_x^a$, and v_x^c, whereat $\dot{v}_x^a$, $\ddot{v}_x^a$ belong to user u_a and v_x^c belongs to user u_c

(b) Two nodes host four VMs that belong to three users that compete for CPU and disk I/O

Figure 5.13: Illustration of the two fairness experiments

vestigates how the FS promotes fairness on one node n_x, when two users u_a and u_c contend for the same PR. User u_a operates two VMs $\dot{v}_x^a$ and $\ddot{v}_x^a$ and user u_c operates one VM v_x^c. All VMs attempt to utilize a maximal amount of CPU time, while not imposing significant load on any other PR.

Without the FS all VMs receives $1/3$ of n_x's CPU time, which is arguably not fair, as both users have the same quota but u_a receives twice the amount of CPU time as u_c. Figure 5.14 shows, how the FS mitigates this unfairness by giving more CPU shares and, thus, more CPU time to the only VM of u_c.

The first three minutes of the experiment's real time demonstrate the FS's flexibility: Within this timeframe only $\dot{v}_x^a$ attempts to utilizes the CPU, which increases u_a's greediness and accordingly decreases the CPU shares of u_a's VMs. Nonetheless, $\dot{v}_x^a$ is able to fully utilize the CPU, because no other VM attempts to utilize it. Only after three minutes, when the other VMs also start utilizing the CPU, the shares take effect and u_a's VMs are throttled in favor of u_c's VM.

These results show that the FS successfully enforces fairness, if necessary, while not decreasing PR utilization/efficiency. In the first three minutes only VM $\dot{v}_x^a$ is active and gets full access to the requested CPU time. In particular, as no other VM is active, constraining $\dot{v}_x^a$'s CPU access would not increase the

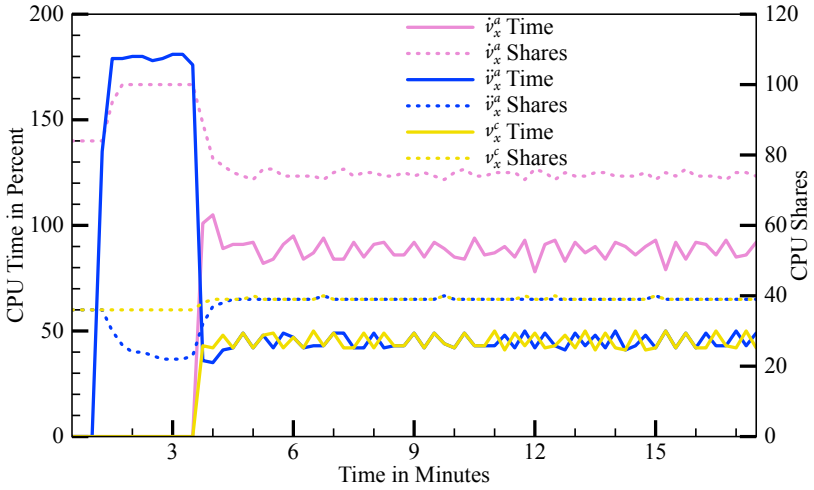


Figure 5.14: CPU shares and resulting CPU time allocated by the FS in the fairness experiment as illustrated in Figure 5.13a

performance of any VM but only decrease the performance of v_x^a. Therefore, constraining v_x^a within the first three minutes would be inefficient. However, after three minutes two other VMs attempt to utilize the CPU and, thus, contention arises. As two of the three VMs contending for CPU time belong to user u_a, it becomes necessary to throttle those two VMs in favor of the third VM, which belongs to user u_c. Otherwise u_a would receive more CPU time than user u_c, which is unfair. Thus, the FS ensures fairness according to Definition 3.6 by prioritizing u_c's only VM.

5.3.3.2 Multiple Nodes, Multiple Resources

The second fairness experiment investigates how the FS promotes fairness across nodes, when users contend for multiple PRs. Figure 5.13b shows this setup, where three users u_a, u_c, and u_e utilize PRs on two nodes n_x and n_z. User u_a heavily utilizes the CPU and disk on n_x by VM v_x^a. User u_c attempts the same PR utilization by VM v_x^c. However, u_c additionally utilizes the CPU of node n_z by VM v_z^c. VM v_z^c shares this node with u_e's only VM v_z^e that heavily utilizes disk. Therefore, the utilization of v_z^c and v_z^e on n_z does not interfere, while v_x^a and v_x^c contend for CPU and disk I/O on n_x.

Figures 5.15 and 5.16 compare CPU and disk allocations with and without the FS on both nodes. The PRs user u_c's VMs receive are illustrated by blue lines in all figures. PRs allocated to user u_a and u_e VMs are illustrated by green and yellow lines, respectively.

Figure 5.15 compares the allocation on n_x. Figure 5.15a demonstrates that without the FS both VMs on n_x receive the same amount of CPU and disk I/O. Figure 5.15b shows that the FS decreases v_x^c's CPU and disk I/O access in order to allocate more of these PRs to v_x^a. The reason is that the owner of v_x^c also consumes PRs on n_z. The allocation on n_z is compared by Figure 5.16 (v_z^c does not utilize the disk, wherefore the according line is not visible in these figures) and shows that the FS increases v_z^e's disk access by almost 40%. The reason is that v_z^c's owner also utilizes PRs on n_x and, therefore, v_z^e is prioritized. Interestingly, although the FS increases v_z^e's disk access by 40%, this does not hurt v_z^c.

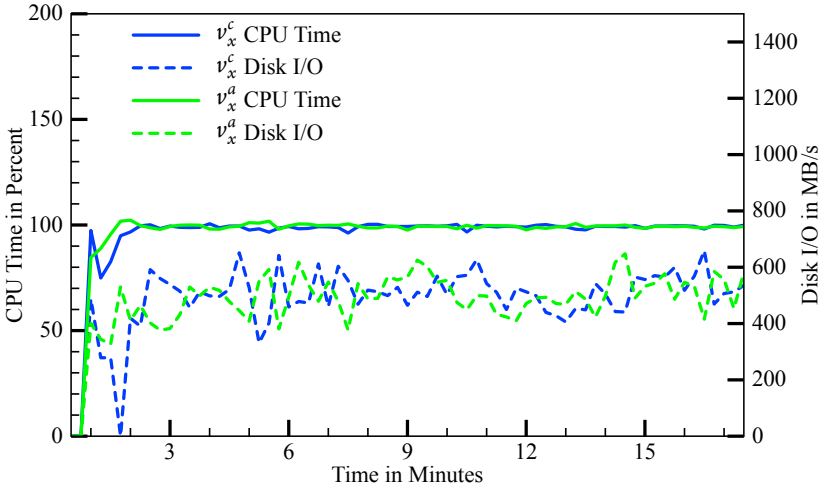
5.3.4 Experimental Results

The FS's CPU overhead is mainly caused by an inefficient implementation for controlling network access. It was investigates in detail how this overhead can be reduced and an according solution was implemented.

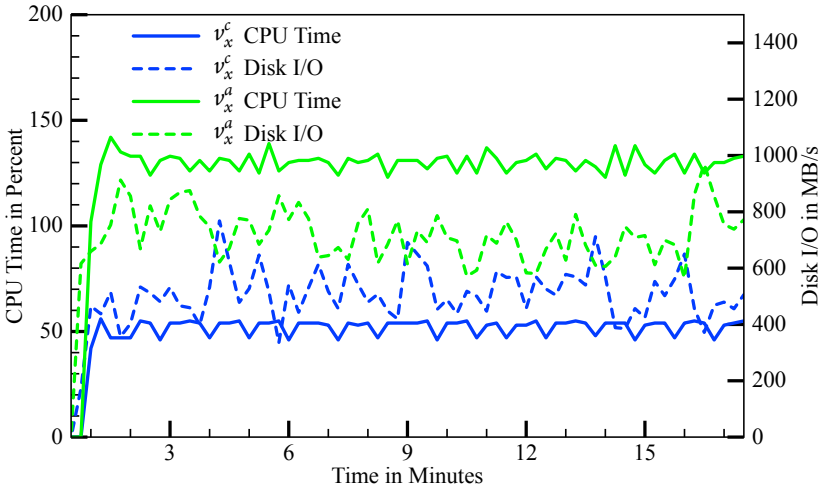
Without the FS, OpenStack does not leverage PR allocation to promote fairness among cloud users. The simple scenario where two users compete for one PR on the same node, most clearly shows this shortcoming (cf. Section 5.3.3.1). Notably, in this scenario fairness can be established without global monitoring information or a multi-resource fairness definition. The FS not only achieves fairness in this scenario, but also in complex settings, where VMs running on different nodes and utilizing different PRs have to be managed (cf. Section 5.3.3.2).

5.4 Structural Investigations

The definitions in Section 3.3.3 and implementation in Section 4.2.4.1 allow to calculate the greediness in a decentralized manner [6, 100]. The resulting message volume for this calculation equals the sum of the size of messages

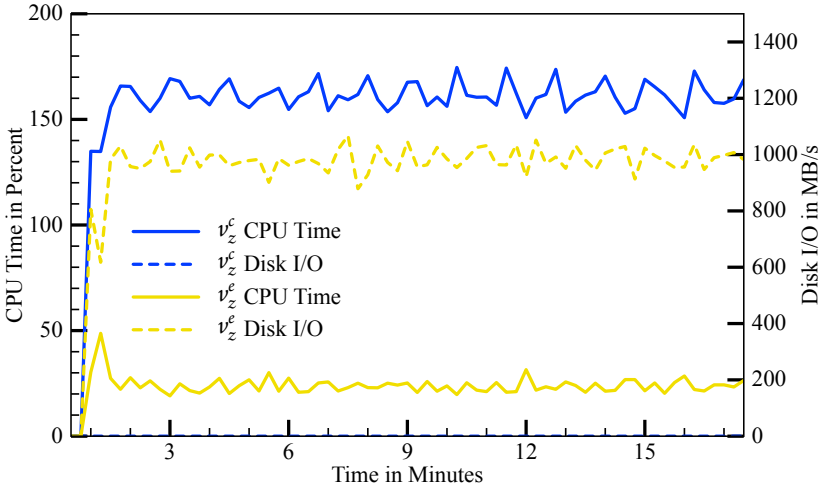


(a) When the FS is inactive

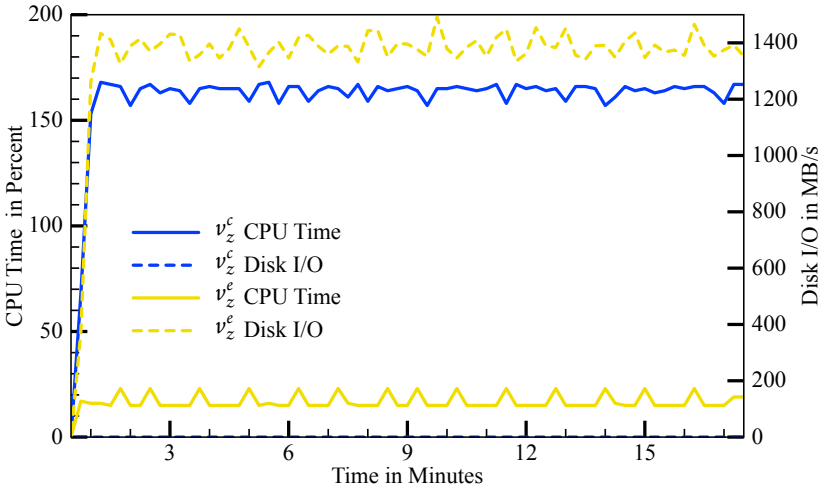


(b) When the FS is active

Figure 5.15: CPU and disk allocation on node n_x of the fairness experiment as illustrated in Figure 5.13b



(a) When the FS is inactive



(b) When the FS is active

Figure 5.16: CPU and disk allocation on node n_z of the fairness experiment as illustrated in Figure 5.13b

exchanged. It is assumed that (a) $\mathfrak{r}$ is small and constant, (b) that

$$\mathfrak{v} \gg \mathfrak{n} \text{ and } \mathfrak{v} \gg \mathfrak{u}, \tag{5.15}$$

because users usually own several VMs and nodes usually host several VMs, and (c) $\forall n \in N\colon |\mathsf{host}(n)| > 0$, *i.e.*, every node hosts at least one VM.

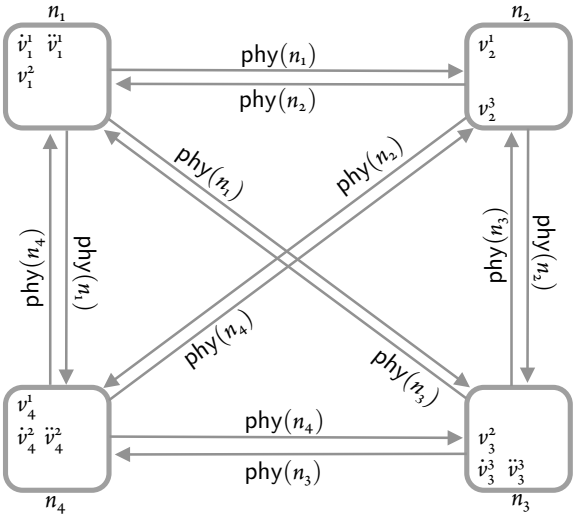
Subsequently, two messaging schemes to calculate the greediness are presented. The messaging scheme presented in Section 5.4.1 is straight-forward but less optimal in terms of message volume then the messaging scheme presented in Section 5.4.2.

5.4.1 Simple Messaging Scheme

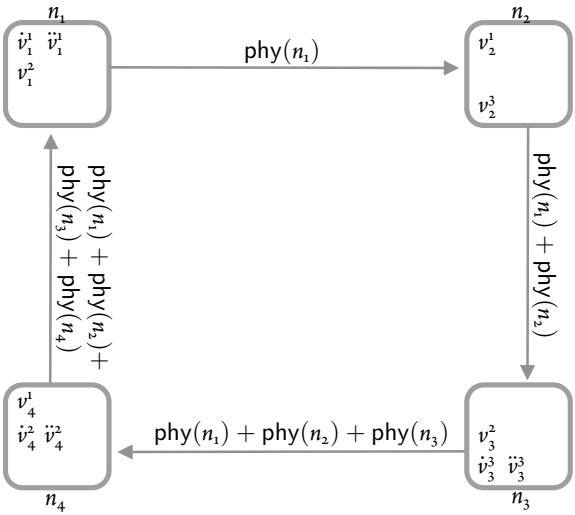
The simple messaging scheme works as follows. Each node applies function G_V to calculate the greediness of the hosted VMs, announces this *greediness set* to all other nodes and accordingly receives the greediness sets of other nodes. Each node then applies function G_U to the calculated and received VM greediness sets to calculate the greediness of users (cf. Listing 4.2).

The CRS is essential to calculate the greediness of VMs and users. As the CRS is the sum of nodes' PRs, every node needs to send its PRs to every other node, such that every node can calculate the CRS. This messaging scheme is illustrated in Figure 5.17a. It is sufficient, to perform this PR announcement once, as the PRs of a node are static (cf. Line 1 and 2 of Listing 4.2). In case a node is added to the cloud, it announces its PRs to every node, to which every node responds with its PRs. Accordingly, each node once has to send a vector $v \in \mathbb{R}^{\mathfrak{r}}_{\geq 0}$ to every other node. This results in $\mathfrak{n} \cdot (\mathfrak{n} - 1)$ messages with a size of $\mathfrak{r}$ being send. The *CRS message volume*, therefore, is $\mathfrak{n} \cdot (\mathfrak{n} - 1) \cdot \mathfrak{r} \in \mathcal{O}(\mathfrak{n}^2)$.

When a node knows the CRS it can locally calculate the greediness of each hosted VM. The resulting greediness set has to be send to every other node, such that every node can calculate the greediness of each user (cf. Line 7 and 8 of Listing 4.2). This results in $\mathfrak{n} \cdot (\mathfrak{n} - 1)$ messages, as illustrated in Figure 5.18a. . In particular, each node n_i has to send a message m to the $\mathfrak{n} - 1$ other nodes. Message m contains $|\mathsf{host}(n_i)\}|$ scalars, wherefore, the size of m is only bounded by $\mathfrak{v}$. This gives bound $\mathfrak{n} \cdot (\mathfrak{n} - 1) \cdot \mathfrak{v} \in \mathcal{O}(\mathfrak{n}^2 \cdot \mathfrak{v})$. However,

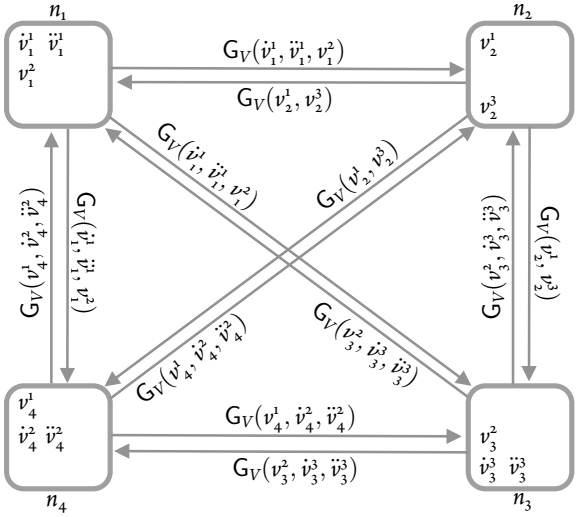


(a) The simple messaging scheme sends messages between each pair of nodes

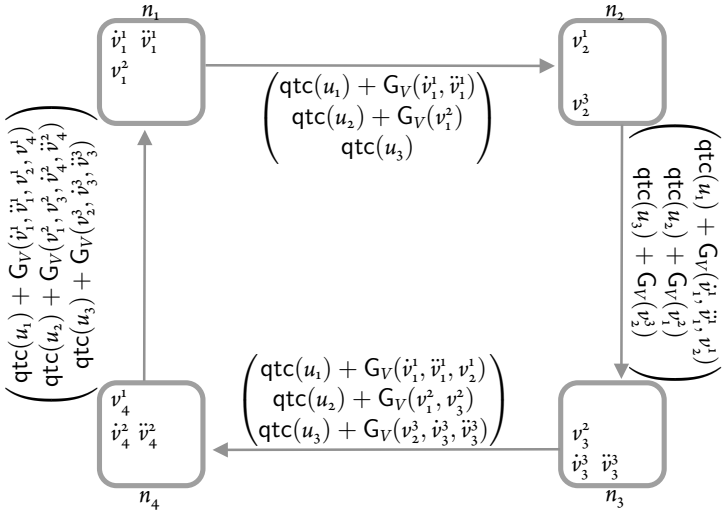


(b) The improved messaging scheme cycles one message through the nodes and every node updates this message

Figure 5.17: An example of the CRS message volume produced by the simple and improved messaging scheme. All message are of size $\mathcal{O}(\mathfrak{r})$

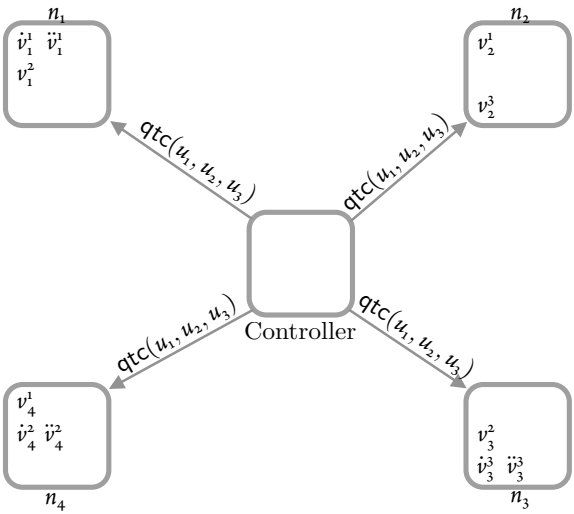


(a) The simple messaging scheme sends the greediness of every VM to every node individually

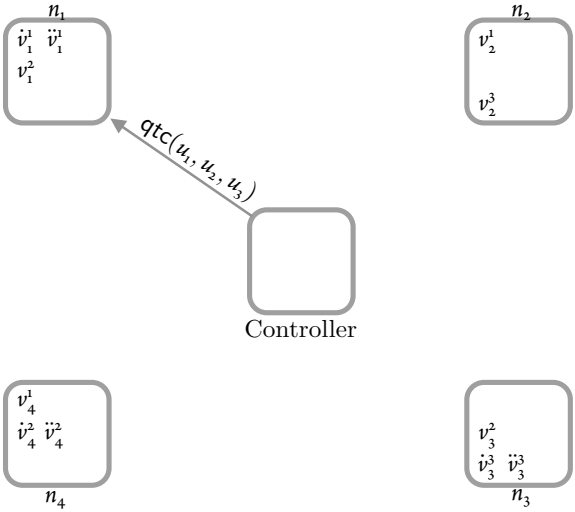


(b) The improved messaging scheme cycles a message of size $\mathcal{O}(u)$ through the nodes

Figure 5.18: An example of the greediness message volume produced by the simple and improved messaging scheme. $\mathsf{G}_V(v_1, v_2, \ldots, v_x)$ abbreviates $\mathsf{G}_V(v_1), \mathsf{G}_V(v_2), \ldots, \mathsf{G}_V(v_x)$



(a) The simple messaging scheme sends the quota credit to every node individually



(b) The improved messaging scheme sends the quota credit to one node, which implicitly communicates it to all other nodes (cf. Figure 5.18b)

Figure 5.19: An example of the quota message volume produced by the simple and improved messaging scheme. $\mathrm{qtc}(u_1, u_2, \ldots, u_x)$ abbreviates $\mathrm{qtc}(u_1), \mathrm{qtc}(u_2), \ldots, \mathrm{qtc}(u_x)$

since the greediness of every VM has to be send to $\mathfrak{n} - 1$ nodes, the *greediness message volume* actually is $(\mathfrak{n} - 1) \cdot \mathfrak{v} \in \mathcal{O}(\mathfrak{n} \cdot \mathfrak{v})$.

In order to calculate the greediness of any user, nodes also need to know the quota credit of every user (cf. Definition 3.4), which is a static scalar for each user. Accordingly, the quota credits must be announced initially to every node by the central entity that holds this information, in the context of OpenStack this node is the controller (cf. Figure 5.19a). Thus, the *quota message volume* is $\mathfrak{n} \cdot \mathfrak{u} \in \mathcal{O}(\mathfrak{n} \cdot \mathfrak{u})$.

As the CRS and quota message volume have to be exchanged once, while the greediness message volume has to be exchanged continually, and because of Equation 5.15, the message volume that needs to be exchanged to calculate and update the greediness of users is $\mathcal{O}(\mathfrak{n} \cdot \mathfrak{v})$. As every node has to be involved in the communication, the first factor $(\mathfrak{n})$ cannot be reduced. The second factor $(\mathfrak{v})$ is possibly large and reflects the amount of data that every node receives periodically. The improved messaging scheme presented next reduces this factor.

5.4.2 Improved Messaging Scheme

The following messaging scheme to calculate the greediness in a decentralized manner has a lower message volume compared to the messaging scheme presented in Section 5.4.1. The message volume is reduced by a vector $m \in \mathbb{R}^{\mathfrak{u}}$ that (a) contains the greediness of all users, (b) cycles through the nodes according to some node order, and (c) is updated by the node that it currently traverses. Without loss of generality, it is assumed that this node order is given by the nodes' indices, *i.e.*, for $1 \leq i < \mathfrak{n}$, node n_i sends m to n_{i+1} and node $n_{\mathfrak{n}}$ sends m to node n_1. Node n_{i+1} and n_1 are called the successor of n_i and $n_{\mathfrak{n}}$, respectively.

m has one entry for every user $u_i \in U$ and this entry is initalized with $-1 \cdot \mathsf{qtc}(u_i)$. Without loss of generality, it is assumed that the ith entry of m corresponds to user u_i. After m has completed the first cycle, this ith entry contains the greediness of user u_i, as every node that hosts VMs of u_i has added the VMs' greediness to the ith entry. From then on, m, which continues to cycle through the nodes, will inform the node it traverses about the greediness of users, while also being updated by that node. To update m, a

node subtracts from m what it added, when it last received m, and adds the current greediness of VMs it hosts to the entries that correspond to the VMs' owners.

More specifically, every node n_j maintains a vector $m_{\text{local}} \in \mathbb{R}^{\mathfrak{u}}$, which stores what n_j added to m in the last round. Therefore, m_{local} is initialized with zeros. When n_j receives m from its predecessor, m is redefined as $m := m - m_{\text{local}}$. Every entry in m_{local} is then overwritten by zero and for each VM $v_k \in \mathsf{host}(n_j)$, $\mathsf{G}_V(v_k)$ is added to the ith entry of m_{local}, whereat i is determined by $\mathsf{own}(v_k) = u_i$. Finally, by again redefining $m := m + m_{\text{local}}$, m contains the greediness of user u_i at the ith entry, for every user $u_i \in U$. m is then send to the successor of n_j. The first iteration of this process is illustrated in Figure 5.18b. As it is the first iteration, nodes do not deduct values.

The cycling of m is initiated by a node, which holds the information about $\mathsf{edw}_U(u_i)$ of every user $u_i \in U$ (in case of OpenStack, this is the Controller). This entity initializes m as

$$m := -1 \cdot (\mathsf{qtc}(u_1), \mathsf{qtc}(u_2), \ldots, \mathsf{qtc}(u_{\mathfrak{u}}))$$

and sends it to an arbitrary node (cf. Figure 5.19b). This node proceeds as if m were received from its predecessor (as m_{local} is initialized with zeros, the initiating node will not subtract anything from the vector but only add the greediness of hosted VMs). This procedure makes it unnecessary to announce $\mathsf{qtc}(u_i)$ of every user u_i to every node individually (cf. Figure 5.18b and 5.19b). Therefore, the quota message volume becomes $1 \cdot \mathfrak{u} \in \mathcal{O}(\mathfrak{u})$.

After m has traversed the circle twice, every node knows the greediness of all users. Therefore, the greediness message volume is reduced to $2 \cdot \mathfrak{n} \cdot \mathfrak{u} \in \mathcal{O}(\mathfrak{n} \cdot \mathfrak{u})$. As $\mathfrak{v} \gg \mathfrak{u}$, this is a much better bound.

The CRS message volume can be reduced by the same technique to $\mathcal{O}(\mathfrak{n} \cdot \mathfrak{r})$. In particular, a node n_i starts by sending vector $m' := \mathsf{phy}(n_i) \in \mathbb{R}^{\mathfrak{r}}$ to its successor n_j. n_j then adds $\mathsf{phy}(n_j) \in \mathbb{R}^{\mathfrak{r}}$ to m' and forwards the result to its successor, and so forth. This process is illustrated in Figure 5.19b. When m' reaches n_i again, it contains the CRS. To inform every node about the CRS, m' has to cycle another round without being modified.

Therefore, in total the message volume is bounded by $\mathcal{O}(n \cdot (u + r))$. Furthermore, every node only receives messages from and sends messages to one other node and the size of this message is bounded by $\mathcal{O}(u + r)$, which ensures a low communication overhead.

5.4.3 Structural Results

The simple messaging scheme is straight-forward but sends messages between all pairs of nodes. Therefore, nodes have process many messages and $\mathcal{O}(n^2)$ messages traverse the network periodically. In contrast, the improved messaging scheme requires every node to only receive messages from and send messages to one other node and the size of this message is bounded by $\mathcal{O}(u + r)$. This ensures a low CPU overhead on individual nodes. This high efficiency is achieved by ordering all nodes on a ring and exchanging all information along this ring by a message that is updated by the node that forwards it. Thus only $\mathcal{O}(n)$ are exchanged periodically making the messaging scheme and, thereby, the FS highly scalable.

5.5 Evaluation Conclusions

The analytical investigation in Section 5.1 highlighted cases in which the G^δ-policy is not influenced by parameter δ and proved that under the assumption of Leontief utility functions the G^δ-policy provides sharing incentive, Pareto efficiency, strategy proofness, and envy freeness. For this assumption also the D-policy, *i.e.*, DRF, which is the de facto standard for fair data center resource allocation, and the C-policy achieve these characteristics.

The simulative investigation in Section 5.2 first pointed out, how parameter δ influences G^δ-allocations and how the G^δ-policy provides incentives to configure VMs correctly, *i.e.*, to choose VMs' VRs such that they are aligned with the VMs' PR utilization. Such incentive is not provided by DRF or any other policy and important, as it indicates which load to expect from VMs and thereby enables scheduling VMs more efficiently. It was demonstrated how unfairness arises among users, when PRs are allocated on a per-VM basis, even when fairness could have been established without information from other nodes or a multi-resource fairness definition (Section 5.3.3.1 con-

firmed these findings by an analog experiment). Lastly, it was shown that a policy with global scope is always superior to its local counterpart and that the G^{δ}-allocation often lies between the C- and D-allocation.

The theoretical investigations in Sections 3.3.4, 5.1, and 5.2 lead to the conclusion that the G^{δ}-policy is superior to the D- and C-policy, as it most intuitive in terms of fairness and provides incentive to users to configure VMs correctly.

The experimental investigation in Section 5.3 first evaluated the developed FS in terms of CPU overhead. It was found that the customized network control leads to high CPU overhead, while without this customization the overhead is moderate. The customization was necessary to control the network access by PPs, as libvirt, which is used to control the other PRs, only allows to assign hard limits to network access. Therefore, libvirt was extended to allow controlling network access by PPs with minimal CPU overhead. This libvirt extension allows all libvirt users and not only the FS to control the network efficiently by PPs. Section 5.3.3.2 showed how the FS coordinated the PR allocation among different nodes and different PRs to increase the degree of fairness.

The structural investigation in Section 5.4 developed two messaging scheme for the FS. The improved messaging scheme presented in Section 5.4.2 allows for exchanging all necessary information among nodes highly efficiently. In particular, this scheme only produces $\mathfrak{n}$ messages periodically, the size of these messages is bounded by $\mathcal{O}(\mathfrak{u} + \mathfrak{r})$ and every node only receives messages from and sends messages to one other node. This ensures a low CPU overhead on individual nodes.

The FS in conjunction with the G^{δ}-policy allows Cloud Service Providers (CSP) to manage their cloud in a simple manner. CSPs only have to create accounts and quotas for their users and the FS ensures that users receive their fair share of the cloud's resources. In particular, greedy users will receive less PRs, if a less greedy user starts to utilize the same PRs. This is necessary to ensure high performance for light users, while greedy users get more resources with less reliability. Different user weights can be taken into account by assigning users different quotas (which results in different quota credits), as the larger a user's quota, the higher the priorities of the user's VMs will be.

6

Summary and Conclusions

Cloud Computing is omnipresent nowadays and deployed by virtually all companies and many private customers. However, commercial cloud providers are not always able to satisfy privacy and performance concerns of customers. Therefore, many companies and institutions also operate a private cloud. In a private cloud, resource allocation is not guided by Service Level Agreements (SLAs) and performance goals are not captured in a machine interpretable manner. Accordingly, nodes treat Virtual Machines (VMs) as processes of equal importance. As users operate different numbers of VMs and these utilize different amounts of Physical Resources (PR), such as CPU time, RAM, disk I/O, and network access, users often receive unequal amounts of these resources.

This thesis improves this situation by showing how to define fairness and enforce this definition in clouds. It was determined how the cloud resource allocation is best controlled, *i.e.*, how cloud resource allocation is changed most effectively. An intuitive, generic definition of fairness was developed

and refined for the cloud context. The practical applicability of this refinement was certified by a prototypical OpenStack implementation.

6.1 General Conclusions

The greediness of consumers can be defined and quantified based on their multi-resource self-servings, as shown by questionnaire among more than 600 individuals. The questionnaire specified real-life scenarios to (a) not constrain the target group in terms of background knowledge and to (b) not distract participants by technical terms and let them fully concentrate on the question of fairness and greediness. Participants were given the option to textually explain the reasoning, which led to their answers. The greediness metric was developed to formalize the most frequent participant reasoning and quantifies the greediness of consumers based on the bundles they have served themselves from a shared resource pool. When applied to the questionnaire scenarios, the greediness metric results in those options that were most frequently selected by participants, which also includes identifying an allocation that maximizes fairness. No other metric, including the cost metric and the metric used by Dominant Resource Fairness (DRF), achieves this favorable result.

A cloud user's user-greediness is defined as the sum of VM-greediness of the user's VMs, whereat VM-greediness is determined by a refinement of the greediness metric. Cloud fairness is defined as the procedure of prioritizing VMs inversely to the user-greediness of their owner. Therefore, cloud fairness works without assumptions on utility functions and is intuitively fair under the premiss that it is fair to constrain greedy consumers in favor of less greedy consumers. Not relying on utility functions is necessary in the cloud context, as this thesis showed that dependencies of PRs and performance and, therefore, utility functions, are (a) highly complex, (b) workload dependent, and sometimes (c) counter-intuitive. These results do not confirm the idealistic assumptions of Leontief or other well defined utility functions, which are typically present in work on data center multi-resource fairness. However, under the common assumption of Leontief utility func-

tions, cloud fairness is strategy prove, Pareto efficient, and provides sharing incentive and envy freeness.

A VM's endowment are the PRs that the cloud budgets for the VM, *i.e.*, the PRs the VM is entitled to, if the PRs were not dynamically shared. A function that calculates the endowment based on a VM's VRs and the VM's host's PRs was developed. The more a VM's PR utilization is aligned with the VM's endowment, the lower the VM's VM-greediness. Therefore, cloud fairness gives incentives to users to configure their VMs as well as possible with the subsequent PR utilization, which makes VRs a good predictor of the VM's upcoming utilization. This allows the Cloud Service Providers (CSP) to place VMs efficiently on nodes.

Cloud fairness is implemented by periodically adapting priorities with which VMs can access their host's PRs during runtime. This priority adaption was identified as more capable than scheduling, which is particularly inefficient to control cloud resources, when VMs run over long periods. The priority adaption is coordinated among nodes, such that all users have access to a fair amount of PRs via their VMs. This approach was prototypically implemented by an OpenStack service called *nova-fairness*, which practically achieves cloud fairness by periodically (a) quantifying the VM-greediness of VMs, (b) aggregating the VM-greediness to calculate user-greediness, and (c) giving priorities to VMs to access the PRs of their host during runtime, whereat the higher the user-greediness of a user, the lower the priorities assigned to the user's VMs. Step (a) and (c) are implemented by use of libvirt, making nova-fairness compatible with a multitude of hypervisors.

This implementation requires a process on every node that hosts VMs and the communication among these processes. This communication deploys the OpenStack message queues and, therefore, can be decentralized by selecting an according message queue. Due to the structure of the definition of VM- and user-greediness, both can be calculated in a distributed manner, such that every node sends and receives messages only from one other node, making nova-fairness highly scalable.

As nova-fairness changes priorities of running VMs, it is complementary to scheduling approaches. However, cloud fairness can also be implemented during scheduling. In particular, [30] proposes a scheduling policy

that achieves DRF by allowing the user with the smallest dominant share to start the next task. By replacing the dominant share metric in this process by the greediness metric, cloud fairness is enforced during scheduling.

6.2 Answers to Research Questions

The conclusions for the five motivating questions introduced in Section 1.2 are shortly summarized below.

Research Question 1: *How to best control cloud resources?*

While it is most common to control resources of private clouds by scheduling, it is inefficient to manage resource allocation, particularly when VMs run over long periods. The reason is that scheduling only allows changing the order in which VMs are started and, therefore, the effectiveness decreases the longer VMs run. Thus, this thesis concludes that cloud resources have to be controlled by changing the PR allocation among running VMs, *i.e.*, changing priorities with which VMs can access PRs of their host during runtime.

While scheduling decisions are often based only on Virtual Resources (VR), this thesis concludes that PR utilization is a much better basis for allocating cloud resources. In particular, a VM can be idle or heavily loaded resulting in very different PR utilization, while the VM's VRs do not change. However, VRs also have to be taken into account, as VMs are placed on nodes, such that high node utilization is achieved, when a VM's VRs are aligned with the VM's PR utilization. Therefore, if a VM's VRs and PR utilization strongly deviate, node utilization is either too high or too low. Thus, allocation decisions have to be based on PR utilization as well as the alignment of VRs and PR utilization of VMs.

Research Question 2: *Which general resource dependencies with respect to VM performance exist when allocating node resources to running VMs?*

Most data center Multi-Resource Allocation (MRA) approaches assume clear and simple dependencies among resources and their combined effect on performance perceived by users. The most prominent example is the assumption of Leontief utility functions when investigating scheduling. The

structure of Leontief utility functions is simple and examples exist why they are insufficient to mode scheduling. Nonetheless, several MRA approaches, including DRF, rely on this assumption.

In order to base the development of a fairness definition on realistic assumptions, this thesis was the first work to investigate dependencies among PRs and their combined effect on VM performance during runtime. However, it had to be concluded that those dependencies are (a) highly complex, (b) workload dependent, and sometimes (c) counter-intuitive. This was shown by benchmarking the performance of VMs depending on different VR configurations and stress on the host system. In particular, it was found that a lack of RAM often not significantly influences VM performance, even if the RAM would be utilized, if it were available. Furthermore, performance of multi-threaded workloads may decrease with the number the VM's Virtual CPUs (VCPUs) and single-threaded workloads may profit, if the VM has multiple VCPUs. This unpredictability prohibits the application of approaches that assume Leontief or other well-defined utility functions and confirms that Leontief utility functions are too idealistic also in this case.

Research Question 3: *How can the greediness of consumers be quantified based on the multi-resource bundles they served themselves from a common resource pool?*

This question arose because of the negative result to Research Question 2. As well-defined utility functions or a well-structured negotiation process could not be assumed, when allocating PRs to running VMs, nodes were conceptualized as "self-serving buffets" for the VMs. Thus, fairness had to be defined by a fair prioritization process that constrains consumers in case of overload. The premiss that it is fair to constrain greedy consumers in favor of less greedy consumers finally led to this research question and was investigated by a questionnaire among more than 600 participants.

The questionnaire revealed that greediness cannot be defined as allocating consumers bundles of equal price, whereat the price of a bundle is defined as the sum of prices of resource units in the bundle and the price of a resource unit is defined based on the overall amount of that resource. One reason for the inadequacy of prices is that they do not capture that the value of a re-

source increases the scarcer it is. Unfortunately, also extending the price to account for the scarcity of resources or the metric used by DRF did not result in a satisfactory greediness metric. Finally, the questionnaire revealed that it is perceived as greedy, when consumers exceed their equal share of resources and not accordingly cede resources to other consumers as compensation. The Greediness Metric (GM) was developed to formally capture this intuitive notion and, therefore, identifies all most common answers to the questionnaire.

Research Question 4: *How can the definition of greediness be refined to define cloud fairness?*

Cloud fairness was defined as the procedure of prioritizing VMs inversely to the greediness of their users. The greediness of a user u was defined as the sum of greediness of u's VMs (subtracted by a number determined by u's quota) and the greediness of a VM v was defined by a Refined GM (RGM). RGM fixed the ratio of GM's only two parameters (these parameters determine how strongly ceding resources is rewarded and how strongly excessive utilization is penalized) and contrasts v's utilization not to the equal share but to a vector e_v that is determined by v's VRs and v's host's PRs.

A numerical analysis showed that, just as GM, RGM identifies all most common answers to the questionnaire (as the questionnaire assumed equality of all consumers, e_v was uniformly defined by the equal share). An analytical investigation proved that v's refined greediness is minimized, when v's utilization perfectly matches e_v. As e_v is determined by v's VRs and v's VRs are configured by v's owner, users have incentive to configure VMs perfectly, in order to minimize their greediness and, thereby, maximize their priority. The analytical analysis also showed that, if users are uncertain about the upcoming utilization of their VMs, they still have incentive to configure their VMs to the best of their knowledge, and that they neither have incentive to artificially partition their workloads to multiple VMs nor to artificially concentrate it on a monolithic VM. Lastly, the analytical investigation showed that using RGM to prioritize VMs provides sharing incentive, Pareto efficiency, strategy proofness, and envy freeness, when Leontief utility functions are assumed. A simulative investigation quantified the greediness of users by

RGM, the price metric, and the dominant resource metric (used by DRF) and prioritized VMs accordingly. The investigation revealed that the allocations generated, when using RGM, lie between the other two allocations. Therefore, RGM is a compromise between two prominent metrics to quantify multi-resource consumption, but the only metric that rewards correct VM configuration. The simulative investigation confirmed that treating VMs independent of their owners leads to unfairness, even in simple cases.

Research Question 5: *How can the new cloud fairness definition be practically enforced, when allocating node resources to running VMs?*

The cloud fairness definition above was implemented for OpenStack by a prototypical nova service called nova-fairness. Every node that hosts VMs runs this service. Nova-fairness deploys libvirt to inquire how many PRs VMs utilize during runtime and to change priorities with which VMs can access these PRs. This use of libvirt makes nova-fairness compatible with most of the known hypervisors [87]. The high CPU overhead of the prototypical nova-fairness implementation was caused by controlling network access in a suboptimal manner. Therefore, libvirt was extended to allow controlling network access by PPs with minimal CPU overhead. This libvirt extension allows all libvirt users and not only the FS to control the network efficiently by PPs. Nova-fairness instances on different nodes exchange messages via OpenStack message queues, which allows to decentralize this message exchange. A messaging scheme to minimize the message overhead was presented. This scheme achieves high efficiency by ordering all nodes on a ring and exchanging all information along this ring by a message that is updated by the node that forwards it. Therefore, every node is required to send and receive messages only from one other node, making nova-fairness highly scalable.

Nova-fairness proves that it is possible to enforce the new cloud fairness definition practically. Thus, nova-fairness not only enforces intuitive fairness but also provides incentive to users to configure VMs correctly. Providing this incentive is advantageous, as the more the VMs' configuration is aligned with their actual consumption, the more efficiently the CSP can place VMs on nodes. Furthermore, nova-fairness highlights the potential of controlling

cloud resources by changing priorities with which VMs can access the PRs of their host during runtime. As discussed in Research Question 1, this approach is highly advantageous compared to controlling cloud resources by VM scheduling, which is the standard approach to control cloud resources.

6.3 Future Work

Future work encompasses three major areas: the improvement of implementation details of nova-fairness, the development of pricing models enabled by nova-fairness, and the investigation of utility functions in data centers. These areas are discussed subsequently.

The following implementation details of nova-fairness leave room for improvement and further research [75]. Firstly, nova-fairness compares CPU time across nodes by the nodes' BogoMIPS [110]. However, BogoMIPS do not define a scientifically reliable measure to compare CPUs, wherefore other normalization references, such as the SPEC value [96], should be considered for future improvements. Secondly, nova-fairness currently uses a linear function to map user-greediness to VM priorities. The precise design of this function and its effects on the behavior of nova-fairness have to be evaluated. In particular, asymptotic functions and tradeoffs between starving VMs and enforcing fairness most vigorously have to be investigated. Thirdly, when a node is overloaded some VMs have to be live-migrated to other nodes, which temporarily decreases their performance. Therefore, nova-fairness should influence these live-migration decisions to migrate VMs of greedy users.

The second area for future work is the investigation of pricing models enabled by nova-fairness. In particular, nowadays it is common practice in commercial clouds that customers pay on a per-VM-basis [3]. However, the telecommunications sector has shown that customers often prefer flat rates [2, 53, 68], even if a volume-based tariff would reduce costs [50]. With a cloud flat rate customers pay a monthly fee to get access to a cloud, where they can start VMs. Just as Internet flat rates come with a maximum bandwidth, cloud flat rate customers would get a certain quota to spawn VMs. The allocation of PRs to VMs then has to be supported by running nova-fairness to ensure that all customers receive a fair share of the cloud's capacity. In par-

ticular, nova-fairness will ensure that greedy customers receive the majority of the cloud's resources on a best-effort basis, while light customers receive requested resources instantly. Such pricing model also has to account for resources that are currently not considered by nova-fairness, *e.g.*, Graphics Processing Units (GPU), disk space, and software licenses.

After no related work on utility functions in data centers was found, this thesis started according investigations. The results achieved were negative, *i.e.*, it was concluded on which structure utility functions in clouds do not have but not which they do have. Therefore, additional research must be conducted on this topic and positive results formulated as a class of utility functions, that appropriately reflects technical realities, while also allowing for an exhaustive theoretical analysis. Establishing a realistic and practical class of utility functions would allow for the development of highly specialized fairness definitions and allocation policies, with high practical applicability. Furthermore, resource allocations could be optimized in terms of efficiency, irrespective of fairness aspects. Therefore, a realistic class of utility functions would greatly aid data center resource allocation not only in terms of fairness. The current lack of work on this topic, despite its value for the research community, may be caused by the topic's complexity.

Bibliography

[1] Adaptive Computing Enterprises, Inc. Moab Workload Manager Administrator Guide. http://docs.adaptivecomputing.com/mwm/archive/6-0/3.2environment.php, 2011. Online; accessed February 8th, 2016.

[2] Jörn Altmann, Björn Rupp, and Pravin Varaiya. Effects of Pricing on Internet User Behavior. *Netnomics*, 3(1):67–84, June 2001.

[3] Amazon Web Services, Inc. Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/, 2017. Online; accessed January 26th, 2017.

[4] Amos Waterland. stress. http://people.seas.harvard.edu/~apw/stress/, 2014. Online; accessed February 10th, 2016.

[5] Simon Balkau. Investigation of Resource Monitoring Capabilities of KVM and OpenStack. Assignment, https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/VA_s_balkau.pdf, Universität Zürich, Zurich, Switzerland, May 2015.

[6] Simon Balkau. Investigation of Message Exchange Capabilities of OpenStack. Bachelor Thesis, https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/BA_s_balkau.pdf, Universität Zürich, Zurich, Switzerland, June 2017.

[7] Ishan Banerjee, Fei Guo, Kiran Tati, and Rajesh Venkatasubramanian. Memory Overcommitment in the ESX Server. *Vmware Technical Journal*, 2(1):2–12, June 2013.

[8] Andrés Baumeler. Investigating RAM Congestion Effects on Virtual Machine Performance and Resource Utilization. Assignment, Universität Zürich, Zurich, Switzerland, April 2016.

[9] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. *Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pp 41–41, Anaheim, CA, USA, April 2005.

[10] Dimitris Bertsimas, Vivek F. Farias, and Nikolaos Trichakis. On the Efficiency-Fairness Trade-off. *Management Science*, 58(12):2234–2250, 2012.

[11] Arka A. Bhattacharya, David Culler, Eric Friedman, Ali Ghodsi, Scott Shenker, and Ion Stoica. Hierarchical Scheduling for Diverse Datacenter Workloads. *4th Annual Symposium on Cloud Computing*, SOCC'13, pp 1–15, Santa Clara, CA, USA, October 2013.

[12] Thomas Bonald and James Roberts. Enhanced Cluster Computing Performance through Proportional Fairness. *Performance Evaluation*, 79:134–145, April 2014.

[13] Thomas Bonald and James Roberts. Multi-Resource Fairness: Objectives, Algorithms and Performance. *2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'15, pp 31–42, Portland, OR, USA, June 2015.

[14] Raouf Boutaba, Lu Cheng, and Qi Zhang. On Cloud Computational Models and the Heterogeneity Challenge. *Journal of Internet Services and Applications*, 3(1):77–86, December 2011.

[15] Steven J. Brams. *Mathematics and Democracy*. Princeton University Press, Princeton, NJ, USA, 2008.

[16] David Breitgand, Zvi Dubitzky, Amir Epstein, Alex Glikson, and Inbar Shapira. SLA-aware Resource Over-commit in an IaaS Cloud. *8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualization Management (SVM)*, pp 73–81, Las Vegas, NV, USA, October 2012.

[17] Wojciech Burakowski, Andrzej Beben, Hans van den Berg, Joost W. Bosman, Gerhard Hasslinger, Attila Kertesz, Steven Latre, Rob D. van der Mei, Tamas Pflanzner, Patrick Poullie, Maciej Sosnowski, Bart Spinnewyn, and Burkhard Stiller. Traffic Management for Cloud Federation. In: Rob D. van der Mei, Hans van den Berg, and Ivan Ganchev (Editors), *Autonomous Control for a Reliable Internet of Services: Methods, Models, Approaches, Techniques, Algorithms and Tools*. Springer, Heidelberg, Germany, 2017.

[18] Javier Celaya and Loris Marchal. A Fair Decentralized Scheduler for Bag-of-Tasks Applications on Desktop Grids. *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGrid'10, pp 538–541, Melbourne, VIC, Australia, May 2010.

[19] Yann Chevaleyre, Paul E. Dunne, Ulle Endriss, Jérôme Lang, Michel Lemaître, Nicolas Maudet, Julian Padget, Steve Phelps, Juan A. Rodríguez-aguilar, and Paulo Sousa. Issues in Multiagent Resource Allocation. *Informatica*, 30(1):3–31, 2006.

[20] Citrix Systems, Inc. AMQP and Nova. `http://docs.openstack.org/developer/nova/rpc.html`, 2010. Online; accessed February 5th, 2016.

[21] Richard Cole, Vasilis Gkatzelis, and Gagan Goel. Mechanism Design for Fair Division: Allocating Divisible Items Without Payments. *14th ACM Conference on Electronic Commerce*, EC'13, pp 251–268, Philadelphia, PA, USA, June 2013.

[22] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Efficient Datacenter Resource Utilization Through Cloud Resource Overcommitment. *2015 IEEE Conference on Computer Communications Workshops*, INFOCOM WKSHPS, pp 330–335, April 2015.

[23] Danny Dolev, Dror G. Feitelson, Joseph Y. Halpern, Raz Kupferman, and Nathan Linial. No Justified Complaints: On Fair Sharing of Multiple Resources. *3rd Innovations in Theoretical Computer Science Conference*, ITCS'12, pp 68–75, Cambridge, MA, USA, January 2012.

[24] Yoav Etsion, Tal Ben-Nun, and Dror G. Feitelson. A Global Scheduling Framework for Virtualization Environments. *2009 IEEE International Symposium on Parallel Distributed Processing*, IPDPS'09, pp 1–8, Rome, Italy, May 2009.

[25] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An Updated Performance Comparison of Virtual Machines and Linux Containers. Technical report, IBM Research Division, Austin Research Laboratory, 11501 Burnet Road Austin, TX 78758, USA, July 2014.

[26] Donald F. Ferguson, Christos Nikolaou, Jakka Sairamesh, and Yechiam Yemini. Economic Models for Allocating Resources in Computer Systems. In: Scott Clearwater (Editor), *Market Based Control of Distributed Systems*, pp 156–183. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996.

[27] Ed. Floyd, Sally. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166, IETF, Berkeley, CA, USA, March 2008.

[28] Eric Friedman, Ali Ghodsi, and Christos-Alexandros Psomas. Strategyproof Allocation of Discrete Jobs on Multiple Machines. *15th ACM Conference on*

Economics and Computation, EC'14, pp 529–546, Palo Alto, CA, USA, June 2014.

[29] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. Multi-resource Fair Queueing for Packet Processing. *ACM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM 2012, pp 1–12, Helsinki, Finland, August 2012.

[30] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. *8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pp 323–336, Boston, MA, USA, March 2011.

[31] Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. Choosy: Max-min Fair Sharing for Datacenter Jobs with Constraints. *8th ACM European Conference on Computer Systems*, EuroSys '13, pp 365–378, Prague, Czech Republic, April 2013.

[32] Robert P. Goldberg. Survey of Virtual Machine Research. *Computer*, 7(9):34–45, September 1974.

[33] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource Packing for Cluster Schedulers. *2014 ACM Conference on Special Interest Group on Data Communications*, SIGCOMM'14, pp 455–466, Chicago, IL, USA, August 2014.

[34] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource Packing for Cluster Schedulers. *ACM SIGCOMM Computer Communication Review*, 44(4):455–466, October 2014.

[35] Andreas Gruhler. Investigation of Resource Reallocation Capabilities of KVM and OpenStack. Bachelor Thesis, `https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/BA_a_gruhler.pdf`, Universität Zürich, Zurich, Switzerland, August 2015.

[36] Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. Navigating Heterogeneous Processors with Market Mechanisms. *IEEE 19th International Symposium on High Performance Computer Architecture*, HPCA'13, pp 95–106, Shenzhen, China, February 2013.

[37] Deke Guo, Qingbo Wu, Shanshan Li, Yusong Tan, and Quanyuan Wu. Multi-resource Aware Congestion Control in Data Centers. *2013 International Con-*

ference on Parallel and Distributed Systems, ICPADS'13, pp 669–674, Cambridge, MA, USA, May 2013.

[38] Avital Gutman and Noam Nisan. Fair Allocation without Trade. *11th International Conference on Autonomous Agents and Multiagent Systems*, Vol. 2 of *AAMAS'12*, pp 719–728, Valencia, Spain, June 2012.

[39] Björn Hasselmann. Investigation of Multi-Resource Cloud Simulators. Assignment, `https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/VA_b_hasselmann.pdf`, Universität Zürich, Zurich, Switzerland, August 2015.

[40] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. *8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pp 295–308, Boston, MA, USA, March 2011.

[41] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive Algorithms from Competitive Equilibria: Non-clairvoyant Scheduling Under Polyhedral Constraints. *46th Annual ACM Symposium on Theory of Computing*, STOC'14, pp 313–322, New York, NY, USA, May 2014.

[42] Kevin Jackson. *OpenStack Cloud Computing Cookbook*. Packt Publishing, Birmingham, UK, 2012.

[43] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. Technical Report TR-301, Digital Equipment Corp, Hudson, MA, USA, September 1984.

[44] Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. Multi-resource Allocation: Fairness-efficiency Tradeoffs in a Unifying Framework. *31st Annual IEEE International Conference on Computer Communications*, INFOCOM 2012, pp 1206–1214, Orlando, FL, USA, March 2012.

[45] Ian Kash, Ariel D. Procaccia, and Nisarg Shah. No Agent Left Behind: Dynamic Fair Division of Multiple Resources. *2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS'13, pp 351–358, St. Paul, MN, USA, May 2013.

[46] Frank P. Kelly, Aman K. Maulloo, and David K.H. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Sta-

bility. *The Journal of the Operational Research Society*, 49(3):237–252, March 1998.

[47] Dalibor Klusáček and Hana Rudová. Multi-resource Aware Fairsharing for Heterogeneous Systems. *18th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP'14, Vol. 8828 of *Lecture Notes in Computer Science*, pp 53–69, Phoenix, AZ, USA, May 2014.

[48] Dalibor Klusáček, Hana Rudová, and Michal Jaroš. Multi Resource Fairness: Problems and Challenges. *17th Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP'13, Vol. 8429 of *Lecture Notes in Computer Science*, pp 81–95, Boston, MA, USA, May 2013.

[49] Beat Kuster. Exhaustive Assembly of Framework Requirements Necessary to Practically Deploy the Greediness Alignment Algorithm (GAA). Master Thesis, `https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/Beat_Kuster_Masterarbeit.pdf`, Universität Zürich, Zurich, Switzerland, March 2014.

[50] Anja Lambrecht and Bernd Skiera. Paying Too Much and Being Happy about It: Existence, Causes, and Consequences of Tariff-Choice Biases. *Journal of Marketing Research*, 43(2):212–223, May 2006.

[51] Tian Lan, David Kao, Mung Chiang, and Ashutosh Sabharwal. An Axiomatic Theory of Fairness in Network Resource Allocation. *29th Annual IEEE International Conference on Computer Communications*, INFOCOM '10, pp 1–9, San Diego, CA, USA, March 2010.

[52] Gunho Lee, Byung-Gon Chun, and Randy H. Katz. Heterogeneity-aware Resource Allocation and Scheduling in the Cloud. *3rd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'11, pp 4–4, Portland, OR, USA, June 2011.

[53] David Levinson and Andrew Odlyzko. Too Expensive to Meter: the Influence of Transaction Costs in Transportation and Communication. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 366(1872):2033–2046, June 2008.

[54] Jin Li and Jingyi Xue. Egalitarian Division Under Leontief Preferences. *Economic Theory*, 54(3):597–622, November 2012.

[55] Weidong Li, Xi Liu, Xiaolu Zhang, and Xuejie Zhang. A Note on Dynamic Fair Division of Multiple Resources. *CoRR*, abs/1509.07935:1–12, December 2015.

[56] Weidong Li, Xi Liu, Xiaolu Zhang, and Xuejie Zhang. Multi-resource Fair Allocation with Bounded Number of Tasks in Cloud Computing Systems. *CoRR*, abs/1410.1255:1–12, February 2015. Withdrawn.

[57] Weidong Li, Xi Liu, Xiaolu Zhang, and Xuejie Zhang. Dynamic Fair Allocation of Multiple Resources with Bounded Number of Tasks in Cloud Computing Systems. *Multiagent and Grid Systems*, 11(4):245–257, 2016.

[58] Xin Li and Chen Qian. Low-Complexity Multi-Resource Packet Scheduling for Network Functions Virtualization. *IEEE Conference on Computer Communications*, INFOCOM'15, pp 1400–1408, Hong Kong, China, May 2015.

[59] Yusen Li, Xueyan Tang, and Wentong Cai. On Dynamic Bin Packing for Resource Allocation in the Cloud. *26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA'14, pp 2–11, Prague, Czech Republic, June 2014.

[60] Haikun Liu and Bingsheng He. Reciprocal Resource Fairness: Towards Cooperative Multiple-Resource Fair Sharing in IaaS Clouds. *IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'14, pp 970–981, New Orleans, LA, USA, November 2014.

[61] Xi Liu, Xiaolu Zhang, Weidong Li, and Xuejie Zhang. Discrete Interior Search Algorithm for Multi-resource Fair Allocation in Heterogeneous Cloud Computing Systems. *12th International Conference on Intelligent Computation,* ICIC'16, Vol. 9771 of *Lecture Notes in Computer Science*, pp 615–626, Lanzhou, China, August 2016.

[62] Xi Liu, Xiaolu Zhang, Xuejie Zhang, and Weidong Li. Dynamic Fair Division of Multiple Resources with Satiable Agents in Cloud Computing Systems. *5th IEEE International Conference on Big Data and Cloud Computing*, BDCloud'15, pp 131–136, Dalian, China, August 2015.

[63] M. Tim Jones. Inside the Linux 2.6 Completely Fair Scheduler. IBM developerWorks. `http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/`, December 2009. Online; accessed May 11, 2016.

[64] Stephan Mannhart. Development and Evaluation of an OpenStack Extension to Enforce Cloud-wide, Multi-resource Fairness during VM Runtime. Bachelor Thesis, `https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/BA_s_mannhart.pdf`, Universität Zürich, Zurich, Switzerland, March 2016.

[65] Herve Moulin. *Fair Division and Collective Welfare*. MIT Press, August 2004.

[66] Simon Müller. Extending libvirt to Allow for Weighted Network Sharing. Assignment, https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/AS_s_mueller.pdf, Universität Zürich, Zurich, Switzerland, April 2016.

[67] Mohit Narang. Extending libvirt to Allow Weight-proportional Network Sharing. Master Thesis, https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/MA_M_Narang.pdf, Universität Zürich, Zurich, Switzerland, April 2017.

[68] Andrew Odlyzko. Internet Pricing and the History of Communications. *Computer Networks*, 36(5-6):493–517, June 2001.

[69] OpenStack Foundation. Nova's Developer Documentation. http://docs.openstack.org/developer/nova/, 2010. Online; accessed April 27th, 2017.

[70] OpenStack Foundation. OpenStack Wiki: ZeroMQ. https://wiki.openstack.org/wiki/ZeroMQ, 2014. Online; accessed February 5th, 2016.

[71] OpenStack Foundation. OpenStack Installation Guide for Ubuntu 14.04. http://docs.openstack.org/juno/install-guide/install/apt/content/index.html, 2015. Online; accessed February 4th, 2016.

[72] OpenStack Foundation. Services, Managers and Drivers. http://docs.openstack.org/developer/nova/services.html, 2016. Online; accessed February 5th, 2016.

[73] OpenStack Project. https://www.openstack.org/, online; accessed: March 31, 2016.

[74] David C. Parkes, Ariel D. Procaccia, and Nisarg Shah. Beyond Dominant Resource Fairness: Extensions, Limitations, and Indivisibilities. *13th ACM Conference on Electronic Commerce*, EC'12, pp 808–825, Valencia, Spain, June 2012.

[75] Riccardo Patanè. Improvement and Evaluation of an OpenStack Extension to Enforce Cloud-wide Multi-resource Fairness during VM Runtime. Master Thesis, https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/MA_r_patane.pdf, Universität Zürich, Zurich, Switzerland, January 2017.

[76] Phoronix Media. Phoronix Test Suite. http://www.phoronix-test-suite.com, 2017. Online; accessed January 18th, 2017.

[77] Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. FairCloud: Sharing the Network in Cloud Computing. *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM'12, pp 187–198, Helsinki, Finland, August 2012.

[78] Patrick Poullie, Thomas Bocek, and Burkhard Stiller. A Survey of the State-of-the-Art in Fair Multi-resource Allocations for Data Centers. To appear in *IEEE Transactions on Network and Service Management*, TNSM'17, 2017.

[79] Patrick Poullie, Stephan Mannhart, and Burkhard Stiller. Defining and Enforcing Fairness Among Cloud Users by Adapting Virtual Machine Priorities During Runtime. Technical Report IFI-2016.04, https://files.ifi.uzh.ch/CSG/staff/poullie/extern/publications/IFI-2016.04.pdf, Universität Zürich, Zurich, Switzerland, March 2016.

[80] Patrick Poullie, Stephan Mannhart, and Burkhard Stiller. Virtual Machine Priority Adaption to Enforce Fairness Among Cloud Users. *12th International Conference on Network and Service Management*, CNSM'16, Montreal, Canada, October 2016.

[81] Patrick Poullie and Burkhard Stiller. Fair Allocation of Multiple Resources Using a Non-monetary Allocation Mechanism. *7th International Conference on Autonomous Infrastructure, Management and Security,* AIMS'13, Vol. 7943 of *Lecture Notes in Computer Science*, pp 45–48, Barcelona, Spain, June 2013.

[82] Patrick Poullie and Burkhard Stiller. Cloud Flat Rates Enabled via Fair Multi-resource Consumption. Technical Report IFI-2015.03 https://files.ifi.uzh.ch/CSG/staff/poullie/extern/publications/IFI-2015.03.pdf, Universität Zürich, Zurich, Switzerland, October 2015.

[83] Patrick Poullie and Burkhard Stiller. Cloud Flat Rates Enabled via Fair Multi-resource Consumption. *10th International Conference on Autonomous Infrastructure, Management and Security,* AIMS'16, Vol. 9701 of *Lecture Notes in Computer Science*, pp 30–44, Munich, Germany, June 2016.

[84] Patrick Poullie and Burkhard Stiller. The Design and Evaluation of a Heaviness Metric for Cloud Fairness and Correct Virtual Machine Configurations. *13th on Conference on the Economics of Grids, Clouds, Systems, and Services*, GECON'16, Vol. 10382 of *Lecture Notes in Computer Science*, pp 173–178, Athens, Greece, September 2016.

[85] Christos-Alexandros Psomas and Jarett Schwartz. Beyond Beyond Dominant Resource Fairness: Indivisible Resource Allocation In Clusters. Technical report, Tech Report Berkeley, 2013.

[86] Bozidar Radunovic and Jean-Yves Le Boudec. A Unified Framework for Max-Min and Min-Max Fairness With Applications. *IEEE/ACM Transactions on Networking*, 15(5):1073–1083, October 2007.

[87] Red Hat, Inc. Libvirt: Internal Drivers. `https://libvirt.org/drivers.html`, 2016. Online; accessed February 5th, 2016.

[88] Redis Labs. Redis. `http://redis.io/`, 2015. Online; accessed February 5th, 2016.

[89] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. *3rd ACM Symposium on Cloud Computing*, SoCC'12, pp 7:1–7:13, San Jose, CA, USA, October 2012.

[90] Alfréd Rényi. On Measures of Information and Entropy. *4th Berkeley Symposium on Mathematics, Statistics and Probability*, pp 547–561, Berkeley, CA, USA, 1960.

[91] Lutz Schubert and Keith Jeffery. Advances in Clouds. Expert group report, Publications Office of the European Union, Luxembourg, 2012.

[92] Selenic Consulting. smem Memory Reporting Tool. `https://www.selenic.com/smem/`. Online; accessed February 7, 2017.

[93] Bikash Sharma, Ramya Prabhakar, Seung-Hwan Lim, Mahmut T. Kandemir, and Chita R. Das. MROrchestrator: A Fine-Grained Resource Orchestration Framework for MapReduce Clusters. *IEEE 5th International Conference on Cloud Computing*, CLOUD'12, pp 1–8, Honolulu, HI, USA, June 2012.

[94] Kaiji Shen, Xiaoying Zheng, Yingwen Song, and Yanqin Bai. Fair Multi-node Multi-resource Allocation and Task Scheduling in Datacenter. *2012 IEEE Cloud Computing Congress*, APCloudCC'12, pp 59–63, Shenzhen, China, November 2012.

[95] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, NY, USA, 2008.

[96] Standard Performance Evaluation Corporation. Benchmarks. `http://spec.org/benchmarks.html`, 2015. Online; accessed February 20th, 2016.

[97] Patrick Taddei. Design and Development of a CloudSim Module to Model and Evaluate Multi-resource Dependencies. Bachelor Thesis, `https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/BA_p_taddei.pdf`, Universität Zürich, Zurich, Switzerland, September 2015.

[98] Jian Tan, Li Zhang, Min Li, and Yandong Wang. Multi-resource Fair Sharing for Multiclass Workflows. *ACM SIGMETRICS Performance Evaluation Review*, 42(4):31–37, March 2015.

[99] Chandra Thimmannagari. *CPU Design: Answers to Frequently Asked Questions*. Springer, Heidelberg, Germany, 2005.

[100] Bogdan Veres. Further Improvement and Evaluation of an OpenStack Extension to Enforce Cloud-wide Multi-Resource Fairness during VM Runtime. Master Thesis, `https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/MA_b_veres.pdf`, Universität Zürich, Zurich, Switzerland, May 2017.

[101] Hui Wang and Peter Varman. Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-aware Allocation. *12th USENIX Conference on File and Storage Technologies*, FAST'14, pp 229–242, Santa Clara, CA, USA, February 2014.

[102] Wei Wang, Baochun Li, and Ben Liang. Multi-Resource Round Robin: A Low Complexity Packet Scheduler with Dominant Resource Fairness. *21st IEEE International Conference on Network Protocols*, ICNP'13, pp 1–10, Göttingen, Germany, October 2013.

[103] Wei Wang, Baochun Li, and Ben Liang. Dominant Resource Fairness in Cloud Computing Systems with Heterogeneous Servers. *33rd Annual IEEE International Conference on Computer Communications*, INFOCOM'14, pp 583–591, Toronto, Canada, April 2014.

[104] Wei Wang, Baochun Li, Ben Liang, and Jun Li. Multi-Resource Fair Sharing for Datacenter Jobs with Placement Constraints. *International Conference*

for High Performance Computing, Networking, Storage and Analysis, SC'16, pp 1003–1014, Salt Lake City, UT, USA, November 2016.

[105] Wei Wang, Baochun Li, Ben Liang, and Jun Li. Towards Multi-Resource Fair Allocation with Placement Constraints. *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, pp 415–416, Antibes Juan-Les-Pins, France, June 2016.

[106] Wei Wang, Ben Liang, and Baochun Li. Multi-resource Generalized Processor Sharing for Packet Processing. *IEEE/ACM 21st International Symposium on Quality of Service*, IWQoS'13, pp 1–10, Montreal, QC, Canada, June 2013.

[107] Wei Wang, Ben Liang, and Baochun Li. On Fairness-Efficiency Tradeoffs for Multi-resource Packet Processing. *IEEE 33rd International Conference on Distributed Computing Systems Workshops*, ICDCSW'13, pp 244–249, Philadelphia, PA, USA, July 2013.

[108] Wei Wang, Ben Liang, and Baochun Li. Low Complexity Multi-resource Fair Queueing with Bounded Delay. *IEEE Conference on Computer Communications*, INFOCOM'14, pp 1914–1922, Toronto, ON, Canada, April 2014.

[109] Wei Wang, Ben Liang, and Baochun Li. Multi-Resource Fair Allocation in Heterogeneous Cloud Computing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 26(10):2822–2835, October 2015.

[110] Wim van Dorst. BogoMips mini-HowTo. `http://www.clifton.nl/index.html?bogomips.html`, 2006. Online; accessed February 6th, 2016.

[111] Lars-Eric Windhab. Technical Investigation of Computational Resource Interdependencies. Internship, `https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/Lars-Eric_Windhab_Praktikumsbericht.pdf`, Universität Zürich, Zurich, Switzerland, September 2014.

[112] Gerhard J. Woeginger. There is no Asymptotic PTAS for Two-dimensional Vector Packing. *Information Processing Letters*, 64(6):293–297, December 1997.

[113] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, and Cesar A. F. De Rose. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP'13, pp 233–240, Belfast, Ireland, February 2013.

[114] Seyed Majid Zahedi and Benjamin C. Lee. REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors. *19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2014, pp 145–160, Salt Lake City, UT, USA, March 2014.

[115] Yoel Zeldes and Dror G. Feitelson. On-line Fair Allocations Based on Bottlenecks and Global Priorities. *4th ACM/SPEC International Conference on Performance Engineering*, ICPE'13, pp 229–240, Prague, Czech Republic, April 2013.

[116] Jianhui Zhang, Heng Qi, Deke Guo, Keqiu Li, Wenxin Li, and Yingwei Jin. ATFQ: A Fair and Efficient Packet Scheduling Method in Multi-Resource Environments. *IEEE Transactions on Network and Service Management*, 12(4):605–617, December 2015.

[117] Qinyun Zhu and Jae C. Oh. An Approach to Dominant Resource Fairness in Distributed Environment. *28th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems,* IEA-AIE'15, Vol. 9101 of *Lecture Notes in Computer Science*, pp 141–150, Seoul, South Korea, June 2015.

[118] Moshe Zukerman, Liansheng Tan, Hanwu Wang, and Iradj Ouveysi. Efficiency-fairness Tradeoff in Telecommunications Networks. *IEEE Communications Letters*, 9(7):643–645, July 2005.

Other Author's Publications

[119] Frank Gurski and Patrick Poullie. Interval Routing Schemes for Circular-Arc Graphs. *International Journal of Foundations of Computer Science*, 28(1): 39–60, January 2017.

[120] Fabio V. Hecht, Patrick Poullie, Andrei Vancea, and Burkhard Stiller. Attacks on Internet Names. *Readme Alumni - Das Bulletin der Alumni Wirtschaftsinformatik Universität Zürich*, Alumni Wirtschaftsinformatik Universität Zürich, 28(28):14–15, Zurich, Switzerland, October 2012.

[121] ITU-T: Telecommunication Standardization Sector of ITU. Y.3013: Socio-economic Assessment of Future Networks by Tussle Analysis. *ITU-T Recommendation Y.3013*, Patrick Poullie, Martin Waldburger, Corinna Schmitt, and Burkhard Stiller were the main contributors to and editors of this recommendation, August 2014

[122] Ioanna Papafili, Thomas Bocek, David Hausheer, Patrick Poullie, Jürgen Rückert, Sergios Soursos, George Stamoulis, Burkhard Stiller, and Krzysztof Wajda. SMARTenIT Cloud Traffic Management Approach and Architectural Considerations. *22nd Future Network and MobileSummit,* FuNeMS'13, pp 1–11, Lisbon, Portugal, July 2013.

[123] Patrick Poullie, Corinna Schmitt, and Burkhard Stiller. Designing Future Networks: The Investigation of Socio-economic Awareness by the Tussle Analysis. *Conference on Standards for Communications and Networking,* CSCN, pp 1–6, Tokyo, Japan, October 2015.

Appendix

A.1 Questionnaire

The following email was send in English and German to all employees and students of the University of Zurich on July 9, 2015, as well as shared on Facebook. When clicking the link to the survey, which is below indicated by […], participants could first choose the language of the survey (English and German) and were then presented the questions stated in the three subsequent sections.

Subject: *Evaluation Ihres intuitiven Fairnessverständnisses / Evaluation of Your Intuitive Understanding of Fairness*

*****

Der Versand an die ausgewählten Angehörigen der UZH wurde vom
Rektoratsdienst der Universität Zürich ausdrücklich genehmigt.
*****

English translation see below

[German translation of the text below]
****

Evaluation of Your Intuitive Understanding of Fairness

What is fair?
Who should cede something if there is not enough for everybody?

The fair allocation of multiple goods or resources is one of the most profound problems that humankind faced throughout its history.

The Communication Systems Group CSG of the Department of Informatics IFI explores the concept of fairness within a technological context. In particular, data centers are investigated, as they allow individuals to share different computational resources. In these infrastructures, traditional definitions of fairness cannot be verified, wherefore we are trying to define fairness in a way that is applicable there.

The goal of this survey is to evaluate your understanding of fairness and to compare it to the definitions developed at CSG so far. In order not to bother you with technical terms, the questions describe and illustrate simple real-life scenarios and ask you for your opinion on these. Since fairness is a subjective concept, there are no right or wrong answers, and you can help us by giving explanations for your choices in the designated text boxes.

Link to this survey: [link to the survey]

We would really appreciate it, if you would spend 15 minutes of your time to run through this survey.
No personal data is collected.

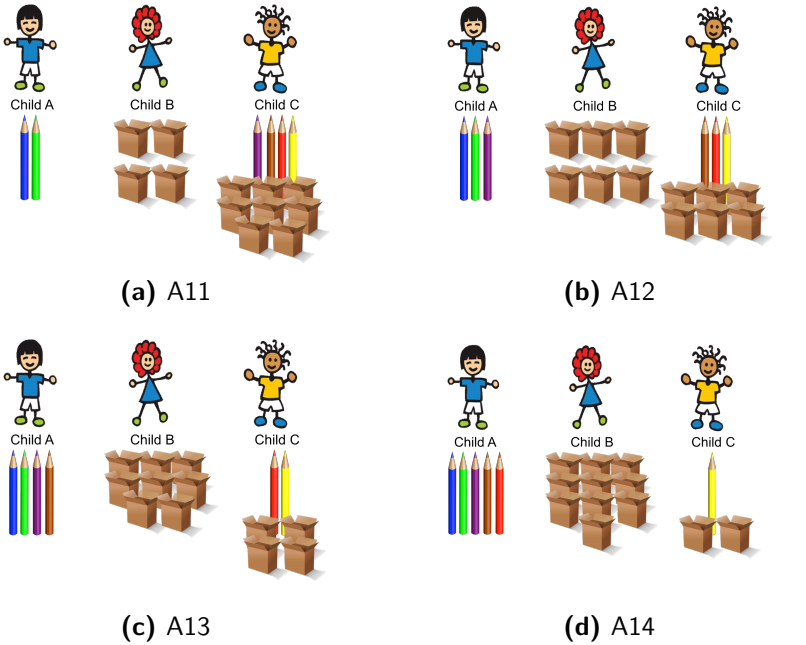
Best regards
Patrick Gwydion Poullie
Institut für Informatik
Communication Systems Group

A.1.1 Choosing the Most Fair Allocation (Q1)

Figure A.1 shows the illustrations of A11, A12, A13, and A14 displayed to participants in Q1 of the questionnaire. The question was phrased as follows:

Three children have six colored pencils and twelve cardboard boxes to play with. Because they do not get along well, their nursery teachers decide that they should play on their own, wherefore the painting materials have to be divided among them while taking into account the three following preferences.

- *Child A wants to color, wherefore he only wants pencils (as many as possible).*
- *Child B wants to build cardboard houses, wherefore she only wants cardboard boxes (as many as possible).*



(a) A11 **(b)** A12

(c) A13 **(d)** A14

Figure A.1: The illustrations presented in Q1 of the questionnaire

- *Child C wants to color cardboard boxes, wherefore for each pencil he receives, he wants two cardboard boxes to color (Child C wants as many of these sets as possible).*

The nursery teachers are unsure how to allocate the pencils and cardboard boxes among the children and must decide between the following four options.

Here the allocations were displayed by the illustrations shown in Table A.1 and by tables that depicted the allocations numerically.

Which allocation do you think is the fairest?

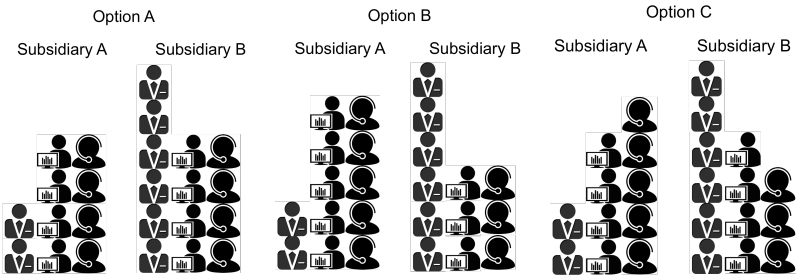
A.1.2 Allocating based on Requests (Q2)

Figure A.3 shows the illustrations of A21, A22, and A23 displayed to participants in Q2 of the questionnaire. The question was phrased as follows:

Human resources of a company are being redistributed. The department heads of two competing subsidiaries A and B must come to an agreement on how 8 account managers, 8 programmers, and 8 administrators are assigned to their de-

Profession	Account Manager	Programmer	Administrator
			
Available	8	8	8
Subsidiary A	2	5	5
Subsidiary B	6	4	4

Figure A.2: The table presented in Q2 of the questionnaire



(a) A21 **(b)** A22 **(c)** A23

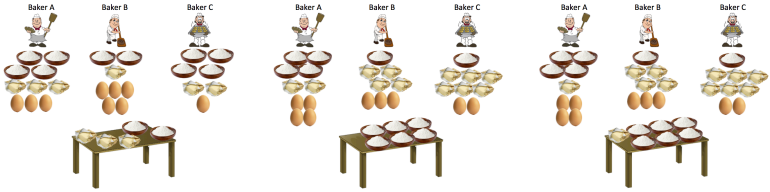
Figure A.3: The illustrations presented in Q2 of the questionnaire

partments. The professional groups' workforce does not suffice to meet the department heads' demands simultaneously, which leads to difficult negotiations. After these negotiations, they have almost reached an agreement as depicted in the following table. In particular, the table shows how many professionals the subsidiaries demand after the negotiations. Initially, the demands for accountant managers also exceeded their availability, but this issue was resolved during the negotiations. Here the Table in Figure A.2 was shown.

Since there is still one programmer and one administrator missing to implement the demands, the corporate management has to decide which of the subsidiaries gets one programmer and one administrator less than requested. Since the two subsidiaries are competing, the management wants to find a fair solution.

What do you think is the fairest solution?

- *Subsidiary A gets one programmer and one administrator less. (Option A)*
- *Subsidiary B gets one programmer and one administrator less. (Option B)*
- *One subsidiary gets one programmer less, the other gets one administrator less. (Option C)*



(a) S31 **(b)** S32 **(c)** S33

Figure A.4: The illustrations presented in Q3 of the questionnaire

Here the illustrations in Figure A.3 were shown.

A.1.3 Estimating Greediness (Q3)

Figure A.4 shows the illustrations of S31, S32, and S33 displayed to participants in Q3 of the questionnaire. The question was phrased as follows:

Three bakers each want to bake a cake. To save money, they buy the ingredients (eggs, butter, and flour) together. However, because each baker is using a different recipe, each baker needs a different amount of ingredients.

To split the ingredients, they put the ingredients on a table and each baker helps himself. After a while at least one ingredient is depleted, wherefore the bakers cannot use more of the remaining ingredients; in other words, the ingredients left on the table cannot be utilized due to the lack of depleted ingredients.

In the following, three scenarios are depicted with what the bakers took from the table and what was available initially. Each scenario is depicted by a table and a graphic, which both demonstrate how much the bakers took of each ingredient.

Rank the three bakers according to how you perceive their greediness based on the amounts they have used.

Here the allocations were displayed by the illustrations shown in Table A.4 and by tables that depicted the allocations numerically.

List of Figures

List of Tables

Acknowledgments

I AM VERY GRATEFUL to a number of people who have helped me, directly or indirectly, with my work on this thesis. Without their continuous support, this work would not have been possible.

First and foremost I would like to thank Prof. Dr. Burkhard Stiller for the liberty to work on this topic, for seeing the big picture and motivating me, when I felt the selected topic was too challenging, as well as for the technical feedback he provided despite all his obligations. I would also like to thank my co-referents, Dr. Ioanna Papafili and Prof. Dr. Georgios Stamoulis for their valuable feedback.

I thank all CSG members for complementing my work with their practical expertise, providing their valuable opinion and objective feedback, managing the physical infrastructure that was used to run many of the simulations and experiments presented in this thesis, and for being great people that I also enjoyed meeting outside of work. Especially I want to thank (in alphabetical order), Dr. Thomas Bocek, Dr. Andrei Lareida, Dr. Guilherme Sperb Machado, Dr. Corinna Schmitt, Dr. Christos Tsiaras, and Dr. Martin Waldburger.

I want to thank the many students, who contributed to this thesis. Of these students, special thanks go to Beat Kuster, for helping me kickstart the work on my thesis, Lars-Eric Windhab, for providing the starting point for the VM benchmarks, Patrick Taddei, for helping me develop the simulator, Andreas Gruhler and Simon Balkau, for investigating different aspects of OpenStack, Stephan Mannhart, for implementing the fairness service, Simon Müller, for comparing different queuing disciplines, and Mohit Narang, for extending Simon's comparison and integrating a queuing discipline into libvirt.

I would like to thank Laura Mascarell Espuny, my parents, brother, and all my friends for all their support over the past years.

Finally, I want to thank all those people, who enrich the Internet with their free, high quality content that made working on this thesis significantly more efficient and pleasant.

Curriculum Vitae

Patrick Gwydion Poullie was born in Düsseldorf, North Rhine-Westphalia, Germany, on December 29, 1986. From 2006 to 2011 he studied at the University of Düsseldorf from which he received his bachelor's and master's degree in computer science, both with distinction. In 2010 he received a grant for excellent students from University of Düsseldorf. During his studies he worked at the chair for bioinformatics and algorithms and data structures. His diploma thesis proves that interval routing schemes work well for circular arc graphs.

From 2011 to 2017 he was a Ph.D. student and Junior Researcher at the Communication Systems Group CSG, Department of Informatics IfI, at the University of Zurich UZH. During this time he has been involved in the European Union projects "Socio-Economic Services for European Research Projects (SESERV)", "Socially-aware Management of New Overlay Application Traffic combined with Energy Efficiency in the Internet (SmartenIT)", and "FLAMINGO", which focuses on network and service management. Also, he was rapporteur at the ITU-T. Due to these activities and the work on his dissertation, he gained technical knowledge, such as monitoring the physical resource utilization of virtual machines, theoretical skills, such as modeling and simulating cloud resource contention, managerial competence, such as leading and coordinating activities of several students, and political experience, such as contributing to and advancing of ITU-T recommendations.

His main research interests are cloud computing, fairness, and graph and complexity theory. His doctoral thesis was supervised by Prof. Dr. Burkhard Stiller (University of Zurich, Switzerland), Dr. Ioanna Papafili (Athens University of Economics and Business), and Prof. Dr. Georgios Stamoulis (Athens University of Economics and Business). He is the first author of six peer-reviewed publications, of which five are covered in his dissertation and one received a best paper award.